HW7+8

April 5, 2022

0.1 HW 7: Data Description & Preprocessing with Input Data Visualization

0.1.1 OCEN 460

0.1.2 Team: __/Sample__Text/

0.1.3 Members: Nate Baker and James Frizzell

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import pathlib
import os

%matplotlib inline
#Github: https://github.com/jafrizzell/coral-prediction.git
#Describe Datasets and project idea
```

- The World Ocean Atlas (WOA) data cannot be shown in it's raw for here because it is too large to be shared in teh github repository where the data is stored. The metadata for it looks as follows. Replace temperature with "salinity" or "dissolved oxygen" for the other two datasets collected from WAO.

Latitude | Longitude | Temperature@0m depth (Celsius) | Temp@5m | Temp@10m | Temp@15m |…| Temp@5500m

- The Deep Sea Coral Data (DSC) has the following metadata

Latitude | Longitude | Depth (m) |

0.1.4 1. The deep sea coral dataset reports latitude and longitude of known coral growth locations with the depth at which the coral is growing. The World Ocean Atlas reports depth measurements in increments of 5 meters for depths of 0 to 100 meters, 10 meters for 100 to 500 meters, 50 meters for 500 to 2000 meters, and in 100 meters for greater than 2000 meters. The following code was used and adjusted to round the Deep Sea Coral dataset to match this convention.

[ ]:
```
path = 'C:/Users/jafri/Documents/GitHub/coral-prediction/processed_data/
 ↪deep_sea_corals_rounded.csv'

raw = pd.read_csv(path)
```

```
def round_depth(x, base):
    return int(base * round(float(x)/base))


raw['depth'] = raw['depth'].apply(lambda x: round_depth(x, base=5))

raw = raw[raw.depth >= 0]
print(len(raw))
raw.to_csv('C:/Users/jafri/Documents/GitHub/coral-prediction/processed_data/
↪deep_sea_corals_rounded_depthcorr.csv')
```

0.1.5 2. The following code aligns latitude and longitude values from the Deep Sea Coral dataset with the lat/long values from the WOA dataset with a tolerance of 0.5 degrees. Second__param file can be changed to indicate the oceanographic variable of interest. WOA data is right-joined to DSC data for further preprocessing.

0.1.6 The code yields a .csv file that contains the DSC data and the WOA data. The WOA data is depth-stratified.

[ ]:
```
import geopandas

coral = 'D:/TAMU Work/TAMU 2022 SPRING/OCEN 460/depthtempsal_short2.csv'
second_param = 'D:/TAMU Work/TAMU 2022 SPRING/OCEN 460/woa18_all_O00mn01.csv' ␣
↪# "O00mn01" indicates O2 data

raw_coral = pd.read_csv(coral)
raw_coral = geopandas.GeoDataFrame(raw_coral, geometry=geopandas.
↪points_from_xy(raw_coral.longitude, raw_coral.latitude))
raw_coral.depth = raw_coral.depth.astype(float)
raw_coral.latitude = raw_coral.latitude.astype(float)
raw_coral.longitude = raw_coral.longitude.astype(float)

raw_param = pd.read_csv(second_param)
raw_param = raw_param.astype(float)
raw_param = geopandas.GeoDataFrame(raw_param, geometry=geopandas.
↪points_from_xy(raw_param.longitude, raw_param.latitude))


depth_sal = raw_coral.sjoin_nearest(raw_param, max_distance=0.5)

depth_sal.to_csv('D:/TAMU Work/TAMU 2022 SPRING/OCEN 460/depthtempsaloxy.csv',␣
↪index=False)
```

0.1.7 3. To resolve the stratified nature of the WOA data, the following code is used to select the corresponding WOA column for the DSC depth of interest.

```
[ ]: path = 'D:/TAMU Work/TAMU 2022 SPRING/OCEN 460/depthtempsaloxy.csv'

raw = pd.read_csv(path)
raw = raw[raw['depth'] <= 5500]
skipped = 0
for i in range(len(raw)):
    try:
        depth = str(raw['depth'][i])
        raw['oxygen'][i] = raw[depth][i]
    except KeyError:
        skipped+=1
        pass

print("skipped:", skipped)

raw.to_csv('D:/TAMU Work/TAMU 2022 SPRING/OCEN 460/depthtempsaloxy_short.csv',␣
 ↪index=False)
```

0.1.8 4. The following code determines the maximum depth for each lat/long pair in the WOA dataset. These datapoints were then used to create a control dataset describing where coral is not present, in order to compare to the DSC dataset. Code displayed in sections 2 and 3 were used to add the temperature, salinity, and oxygen variables to the control dataset.

```
[ ]: path = 'D:/TAMU Work/TAMU 2022 SPRING/OCEN 460/woa18_decav_t00mn04.csv'

raw = pd.read_csv(path)
depth = []

for i in range(len(raw)):
    for j in range(103):
        curr = raw.iloc[i, -1-j]
        plus = raw.iloc[i, -2-j
        if j == 0 and np.isfinite(curr):
            depth.append(raw.columns[-1])
            break
        elif np.isnan(curr) and np.isfinite(plus):
            depth.append(raw.columns[-2-j])
            break
        elif j == 102:
            depth.append(raw.columns[-1])
print(len(depth))
print(len(raw.latitude))
out = pd.DataFrame({'latitude': raw.latitude,
```

```
                    'longitude': raw.longitude,
                    'depth': depth})

out.to_csv('D:/TAMU Work/TAMU 2022 SPRING/OCEN 460/depths.csv', index=False)
```

Ultimately, the final dataset had the following metadata

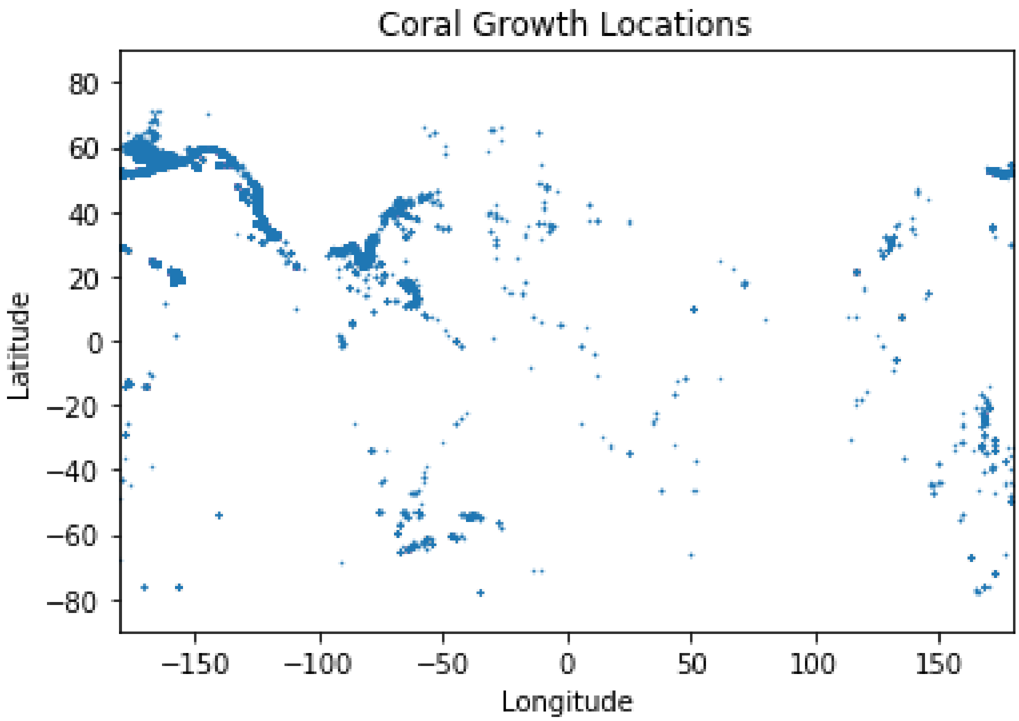
Latitude | Longitude | Depth (m) | Temperature (c) | Salinity (ppt) | Dissolved Oxygen (umol/kg)

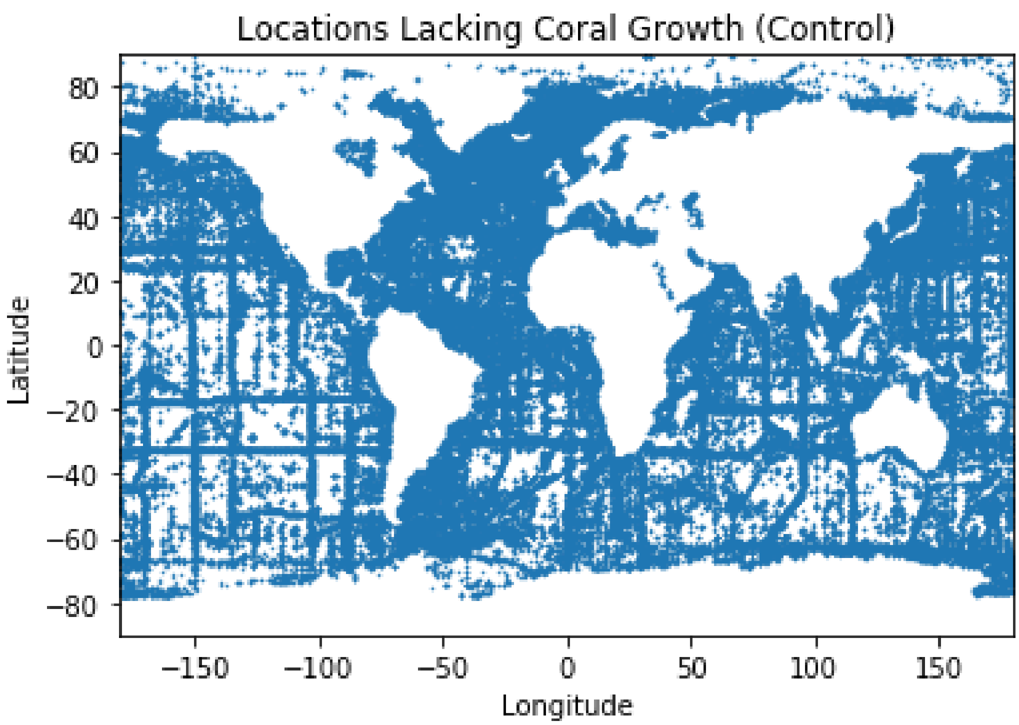
0.1.9 5. The following code visualizes the two datasets.

[2]:
```
#Reprocessed Data for Visualization
path = str(pathlib.Path(os.getcwd())) +␣
 ↪'\processed_data\combined_data_truncated.csv'
raw = pd.read_csv(path)
print(raw.describe())

#Visualization
coral_present_bool = raw[raw.coral_present == 1]
plt.scatter(coral_present_bool['longitude'], coral_present_bool["latitude"], s=␣
 ↪0.2)
plt.title("Coral Growth Locations")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.xlim([-180,180])
plt.ylim([-90,90])
plt.show()

coral_missing_bool = raw[raw.coral_present == 0]
plt.scatter(coral_missing_bool['longitude'], coral_missing_bool["latitude"], s=␣
 ↪0.2)
plt.title("Locations Lacking Coral Growth (Control)")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.xlim([-180,180])
plt.ylim([-90,90])
plt.show()

print("Number of Coral Growth Datapoints:", len(coral_present_bool))
print("Number of Datapoints with no Coral Growth", len(coral_missing_bool))

plt.scatter(raw.longitude, raw.latitude, s=0.2, c=raw.depth)
plt.title("Cumulative Dataset, Colored By Depth")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.xlim([-180,180])
plt.ylim([-90,90])
plt.colorbar()
```

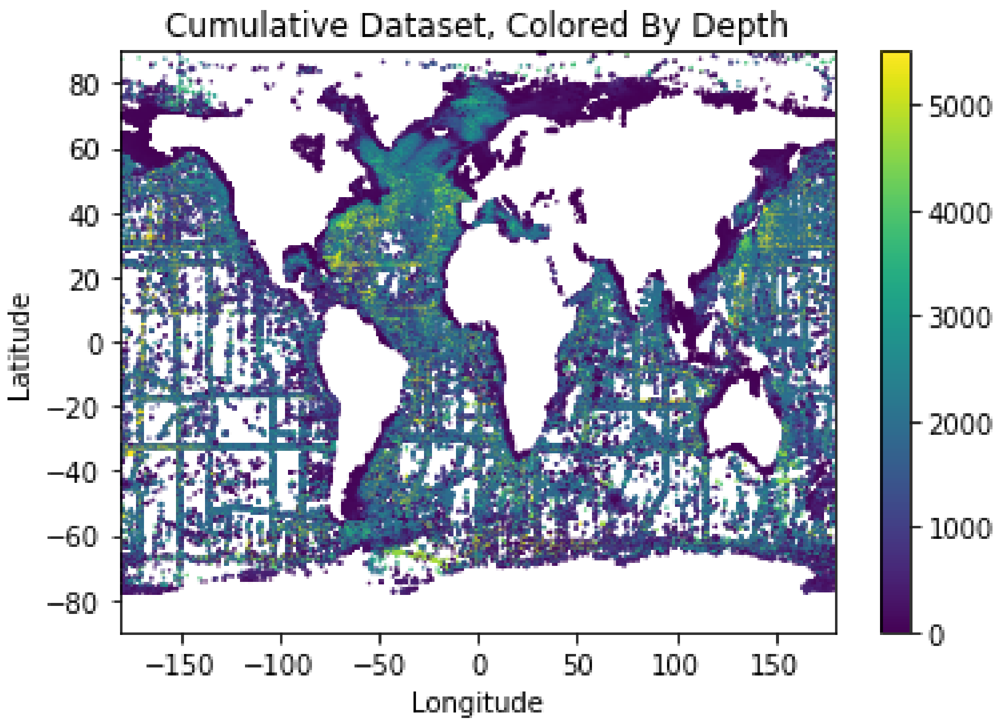
```
plt.show()
plt.scatter(raw.longitude, raw.latitude, s=0.2, c=raw.temperature)
plt.title("Cumulative Dataset, Colored By Temperature")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.xlim([-180,180])
plt.ylim([-90,90])
plt.colorbar()
plt.show()
plt.scatter(raw.longitude, raw.latitude, s=0.2, c=raw.salinity)
plt.title("Cumulative Dataset, Colored By Salinity")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.xlim([-180,180])
plt.ylim([-90,90])
plt.colorbar()
plt.show()
plt.scatter(raw.longitude, raw.latitude, s=0.2, c=raw.oxygen)
plt.title("Cumulative Dataset, Colored By Oxygen")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.xlim([-180,180])
plt.ylim([-90,90])
plt.colorbar()
plt.show()
```

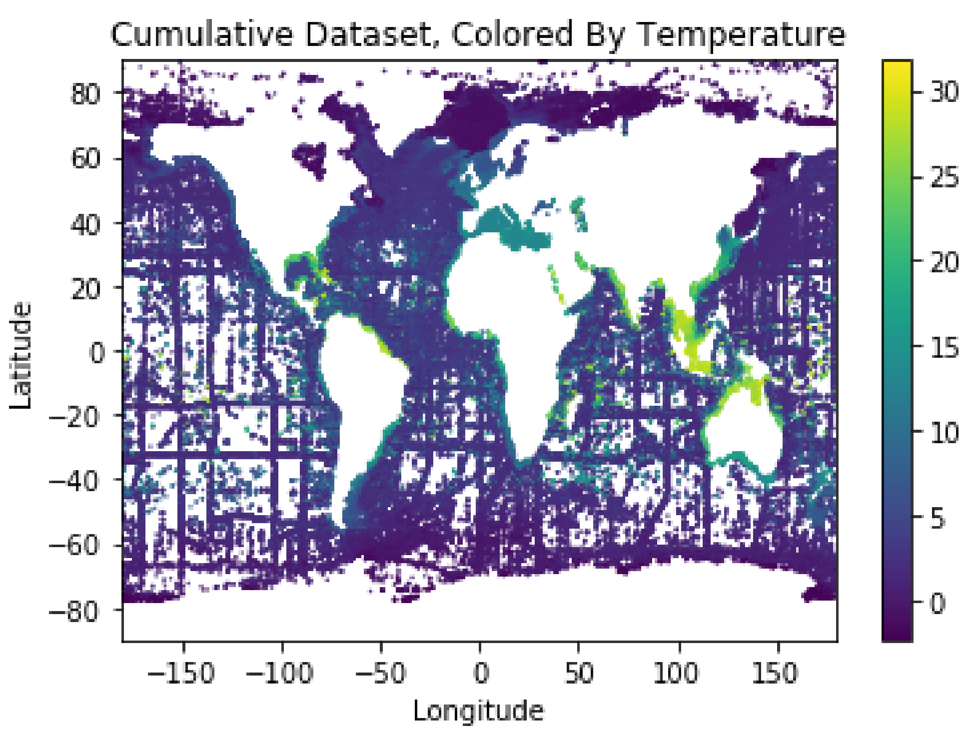
```
       coral_present       latitude      longitude          depth  \
count  341773.000000  341773.000000  341773.000000  341773.000000
mean        0.582340      23.237462     -68.404081    1066.013070
std         0.493174      33.783962      96.027313     989.540361
min         0.000000     -77.875000    -179.989750       0.000000
25%         0.000000      16.625000    -124.339620     235.000000
50%         1.000000      35.641580    -119.498760     850.000000
75%         1.000000      40.811190     -25.625000    1750.000000
max         1.000000      89.875000     179.989750    5500.000000

             round_d    temperature       salinity         oxygen
count  341773.000000  341773.000000  341773.000000  341773.000000
mean     1067.012915       5.191193      34.392171      42.294298
std       989.771161       4.584556       1.343642      29.076344
min         0.000000      -2.271000       0.000000       0.199000
25%       225.000000       2.452000      34.228000      12.890000
50%       850.000000       3.878000      34.520000      41.558000
75%      1750.000000       7.371000      34.699000      65.495000
max      5500.000000      31.751000      41.310000     132.182000
```

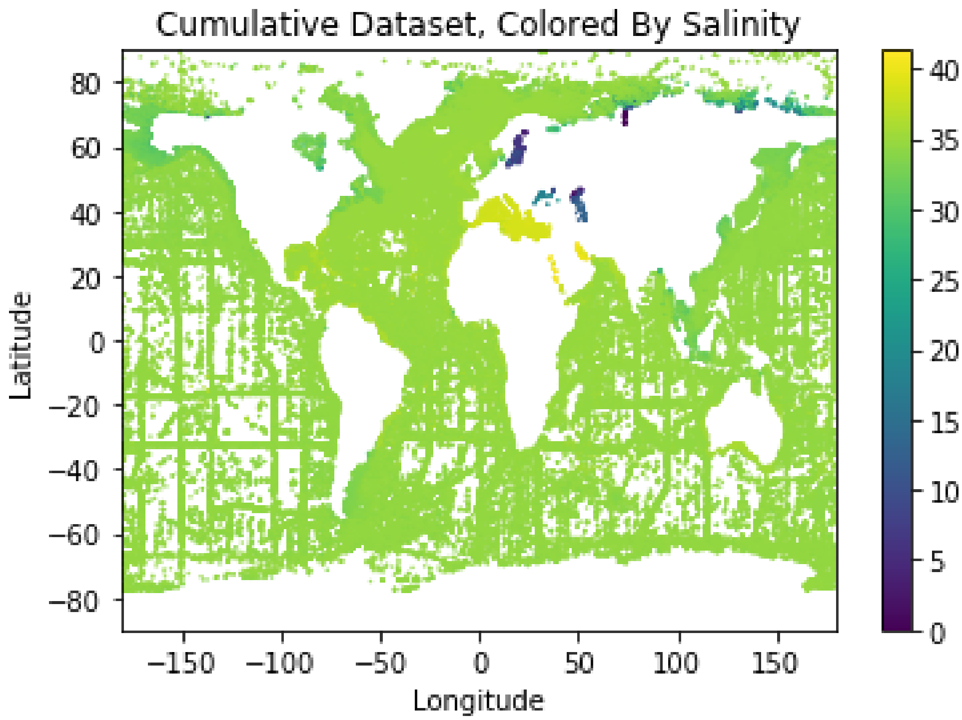

Coral Growth Locations
80
60
40
20
0
−20
−40
−60
−80
Latitude
−150
−100
−50
0
50
100
150
Longitude

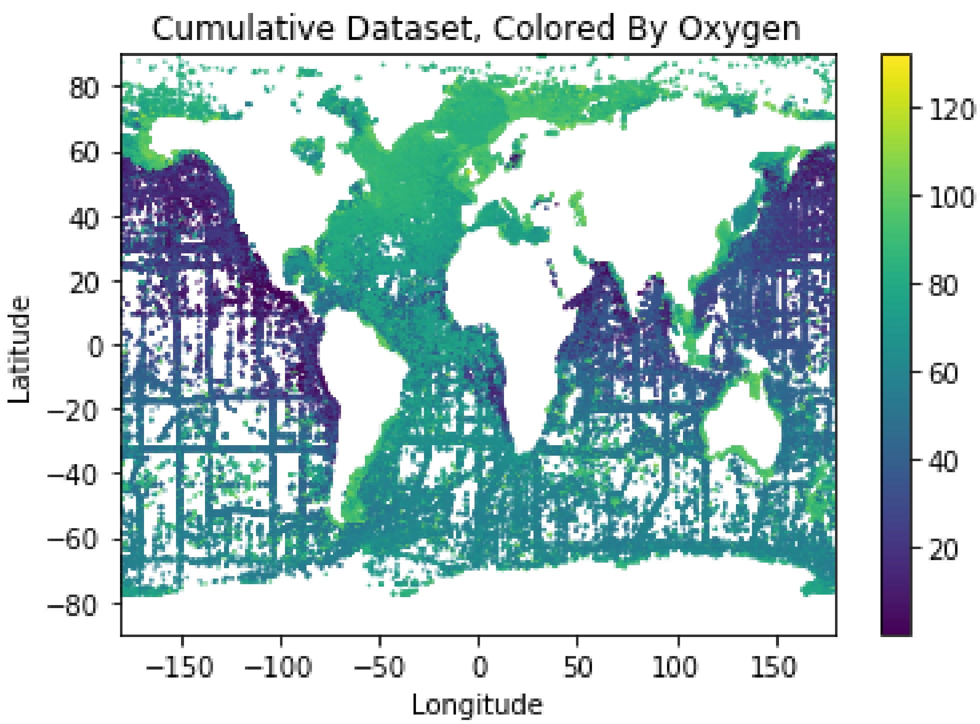

Locations Lacking Coral Growth (Control)
80
60
40
20
0
−20
−40
−60
−80
Latitude
−150
−100
−50
0
50
100
150
Longitude

```
Number of Coral Growth Datapoints: 199028
Number of Datapoints with no Coral Growth 142745
```






Cumulative Dataset, Colored By Salinity
Latitude
Longitude
80
60
40
20
0
−20
−40
−60
−80
−150
−100
−50
0
50
100
150
40
35
30
25
20
15
10
5
0


Cumulative Dataset, Colored By Oxygen
Latitude
Longitude
80
60
40
20
0
−20
−40
−60
−80
−150
−100
−50
0
50
100
150
120
100
80
60
40
20

0.2 HW 8: Pattern Extraction & Why Algorithm Chosen (+ Parameter Tuning)

0.2.1 OCEN 460

0.2.2 Team: __/Sample__Text/

0.2.3 Members: Nate Baker and James Frizzell

Overview:

The purpose of this project is to use existing data on the growth of coral to predict whether coral can grow given oceanographic conditions. The latitude, longitude, depth, temperature, salinity, and dissolved oxygen levels are used to predict a binary value with 1 meaning that coral can grow and 0 meaning that coral cannot grow.

[3]:
```
import os
import tensorflow as tf
import pandas as pd
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
import RegscorePy
from math import sqrt, floor
import pathlib
from itertools import product
from scipy.stats import pearsonr
import time
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
%matplotlib inline
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-3-4b3d92add505> in <module>()
      1 import os
----> 2 import tensorflow as tf
      3 import pandas as pd
      4 import numpy as np
      5 from tensorflow import keras

ModuleNotFoundError: No module named 'tensorflow'
```

Import the necessary packages. Some uncommon ones are RegscorePy, which is a custom open-source library used to calculate the AIC of tensorflow models, and itertools.product which is used to generate all combinations of elements in a list.

0.3 1) Load data in

The raw data is loaded in from a csv file and processed into a training and testing dataset.

[2]:
```
cwd = pathlib.Path(os.getcwd())
path = str(cwd.parent) + '/coral-prediction/processed_data/
 ↪combined_data_truncated.csv'

raw = pd.read_csv(path)


raw = raw.sample(frac=0.2, random_state=0)

print(raw.describe())

raw.pop('species')
raw.pop('round_d')

train = raw.sample(frac=0.8, random_state=0)
test = raw.drop(train.index)

train_features = train.copy()
test_features = test.copy()

train_labels = train_features['coral_present']
test_labels = test_features['coral_present']
train_features.pop('coral_present')
test_features.pop('coral_present')
```

```
       coral_present      latitude     longitude         depth       round_d  \
count   68355.000000  68355.000000  68355.000000  68355.000000  68355.000000
mean        0.584273     23.283085    -68.820676   1063.477800   1064.483798
std         0.492850     33.799299     95.889386    985.059453    985.321543
min         0.000000    -77.875000   -179.966080      0.000000      0.000000
25%         0.000000     16.875000   -124.591595    235.000000    225.000000
50%         1.000000     35.652870   -119.501450    855.000000    850.000000
75%         1.000000     40.811420    -26.375000   1735.000000   1750.000000
max         1.000000     89.625000    179.973800   5500.000000   5500.000000

        temperature      salinity        oxygen
count  68355.000000  68355.000000  68355.000000
mean       5.175744     34.384281     42.198043
std        4.551002      1.371827     29.077581
min       -2.248000      0.022000      0.434000
25%        2.457000     34.228000     12.890000
50%        3.878000     34.520000     41.558000
75%        7.371000     34.696000     65.169500
max       31.751000     41.310000    120.705000
```

[2]:
```
304228    0
213311    0
172619    1
141430    1
133490    1
         ..
267861    0
209754    0
192209    1
36135     1
278318    0
Name: coral_present, Length: 13671, dtype: int64
```

0.4 2) Feature evaluation - correlation coefficients

The correlation coefficients and p-values for each feature are reported. Since all p-values are <0.05, each feature selected is relevant to the model and will be kept.

It is interesting that the longitude is strongly negatively correlated to the presence of coral. This implies that at more eastern locations (say 90E - 180E) it is less likely that coral will grow. This may be a residual of other conditions, such as the much deeper waters in the eastern Pacific Ocean - since the depth is also negatively correlated.

[3]:
```python
print(train.corr()['coral_present'])
print(pearsonr(train_features['latitude'], train_labels))
print(pearsonr(train_features['longitude'], train_labels))
print(pearsonr(train_features['depth'], train_labels))
print(pearsonr(train_features['temperature'], train_labels))
print(pearsonr(train_features['salinity'], train_labels))
print(pearsonr(train_features['oxygen'], train_labels))
```

```
coral_present    1.000000
latitude         0.483094
longitude       -0.623733
depth           -0.381216
temperature      0.244996
salinity        -0.075697
oxygen          -0.484223
Name: coral_present, dtype: float64
(0.48309400646187506, 0.0)
(-0.6237334337918604, 0.0)
(-0.38121618825430753, 0.0)
(0.2449964411644198, 0.0)
(-0.07569703383008823, 2.618925049563251e-70)
(-0.4842228643655442, 0.0)
```

0.5 3) Function definitions

The following functions were used during the training and evaluation of the model.

fit_and_evaluate() accepts a model architecture and fits the training data to it, and the reports the accuracy metrics based on the test dataset.

The hyperparameters selected for the training are: 20% validation split and 50 epochs training duration. These were selected by testing different configurations and choosing the hyperparameters that resulted in the most accurate model.

plot_loss() accepts the model training residuals and plots them over the training duration (number of epochs) the loss and validation loss are both shown.

add_layer() is a part of the parametric model study, which can add hidden layers to a tensorflow model by passing parameters and hyperparameters. This functionality will hopefully be bundled into a package at some point so that users can pip install the ability to do a parametric study.

build_and_compile_model() accepts the model architecture from the user and creates the tensorflow model. It then fits the model and performs the accuracy evaluations.

[4]:
```
def fit_and_evaluate(architecture):
    dnn_model = build_and_compile_model(architecture)

    history = dnn_model.fit(train_features, train_labels, validation_split=0.2,␣
↪verbose=0, epochs=50)
    plot_loss(history)

    test_results = dnn_model.evaluate(test_features, test_labels, verbose=0)
    test_predictions = dnn_model.predict(test_features).flatten()
    r2= r2_score(np.asarray(test_labels).flatten(), test_predictions)
    mae = mean_absolute_error(np.asarray(test_labels).flatten(),␣
↪test_predictions)
    aic = RegscorePy.aic.aic(np.asarray(test_labels, dtype=float).flatten(), np.
↪asarray(test_predictions).astype(float), 4+2)
    rmse = sqrt(mean_squared_error(np.asarray(test_labels).flatten(),␣
↪test_predictions))
    return dnn_model, aic, r2, mae, rmse, test_predictions
```

[5]:
```
def plot_loss(history):
    plt.plot(history.history['loss'], label='loss')
    plt.plot(history.history['val_loss'], label='val_loss')
    # plt.ylim([0, 30])
    plt.xlabel('Epoch')
    plt.ylabel('Error')
    plt.legend()
    plt.grid(True)
    plt.show()
    # pass
```

[6]:
```
def add_layer(dets, hyper, prev):
    default = ['relu']
    try:
```

```
        layer = layers.Dense(dets, activation=hyper[0])(prev)
    except IndexError:
        layer = layers.Dense(dets, activation=default[0])(prev)
    return layer
```

[7]:
```
def build_and_compile_model(arch):
    # Adjust the number of hidden layers and neurons per layer that results in␣
 ↪best fit NN
    hidden_layers = []
    inputs = keras.Input(shape=(6,))
    norm_layer = layers.BatchNormalization()(inputs)
    hidden_layers.append(inputs)
    hidden_layers.append(norm_layer)
    for i in range(num_hidden):
        if arch[i] == 0:
            pass
        else:
            layer = add_layer(arch[i], arch[num_hidden:], hidden_layers[-1])
            hidden_layers.append(layer)
            layer = layers.Dropout(rate=0.2)(hidden_layers[-1])
            hidden_layers.append(layer)
    outputs = layers.Dense(1)(hidden_layers[-1])
    hidden_layers.append(outputs)
    model = keras.Model(inputs=inputs, outputs=outputs)

    model.compile(loss='mean_absolute_error',
                  optimizer=tf.keras.optimizers.Adam(0.001))
    return model
```

0.6 4) Using the model trainer

The following code sets up the necessary data to train the models. By commenting out the line:

list(product(*[l1, l2, l3, activ])), the parametric search is deactivated, and only the architecture specified in the next line is fitted

Additionally, some timing features are implemented. Passing a large parametric space can result in the program running for over 4 hours, training each model. Because of this, a progress meter is added so that the user can see how far along the program is.

[8]:
```
models = []
aic_scores = []
r2_scores = []
maes = []
rmses = []
l1 = np.linspace(32, 256, 5)
l2 = np.linspace(0, 256, 5)
l3 = np.linspace(0, 256, 5)
```

```
activ = ['relu', 'tanh']
# parametric_space = list(product(*[l1, l2, l3, activ]))
parametric_space = [[200, 64, 192, 'relu']]
print(parametric_space)
num_hidden = len(list(i for i in parametric_space[0] if isinstance(i, (int or␣
 ↪float))))
num_hyper = len(parametric_space[0]) - num_hidden
start_t = time.time()
c = 1
```

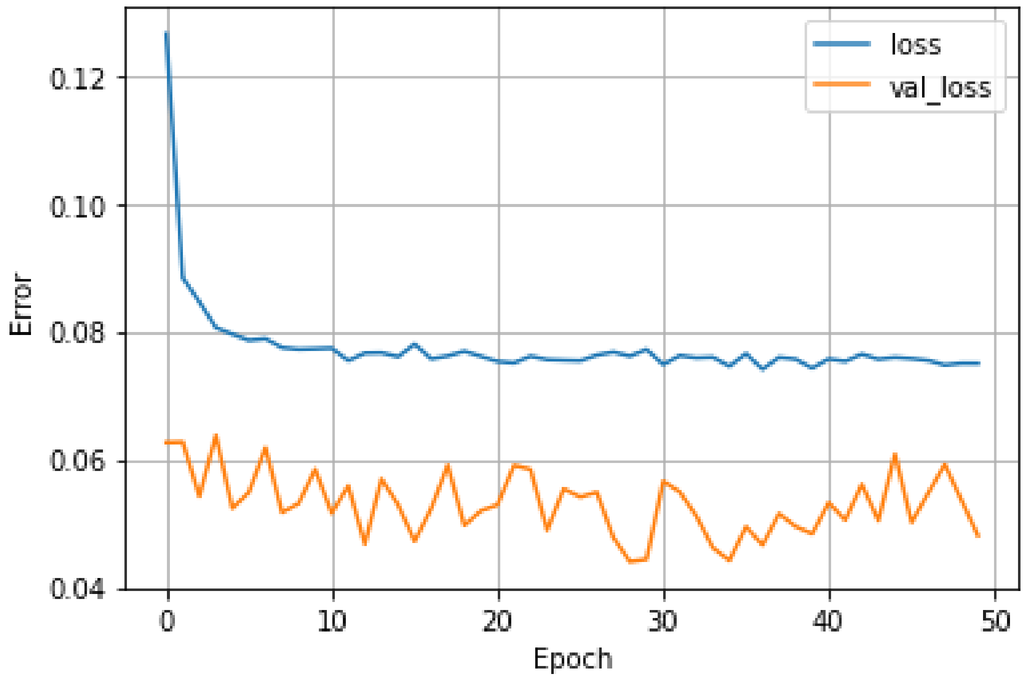
```
[[200, 64, 192, 'relu']]
```

0.7 5) Running the model training

This for loop passes each architecture specified in the parametric space into the build_and_compile_model() function and reports the accuracy metrics into their specific list. It also updates the progress each time a model is finished evaluation

[9]:
```
for arch in parametric_space:
    print('Progress: ' + str(c) + '/' + str(len(parametric_space)))
    dnn_model, aic, r2, mae, rmse, test_predictions = fit_and_evaluate(arch)
    models.append(dnn_model)
    aic_scores.append(aic)
    r2_scores.append(r2)
    maes.append(mae)
    rmses.append(rmse)

    curr_time = time.time()
    diff_t = curr_time - start_t
    t_per_model = diff_t / c
    num_mods_rem = len(parametric_space) - c
    t_rem = t_per_model * num_mods_rem
    print("Estimated Time Remaining: " + time.strftime('%H:%M:%S', time.
 ↪gmtime(t_rem)) + ' seconds')
    c += 1
```

```
Progress: 1/1
```




```
Estimated Time Remaining: 00:00:00 seconds
```

0.8 6) Choosing the best model

After the parametric seach is finished, the accuracy metrics are compiled into a csv file. The user must look through these results and pick the model which has the LOWEST AIC score and HIGHEST R-Squared.

The final 2 lines save the zeroth model in the parametric space for later use. During the parametric seach, this should be disabled because the zeroth model is likely not the most accurate. When the parametric search is disabled (only a single architecture is being fitted) this can be re-enabled to save the model.

[10]:
```
parametric_space_t = np.asarray(parametric_space).transpose().tolist()
output_data = [parametric_space_t[0], parametric_space_t[1],␣
 ↪parametric_space_t[2], aic_scores, maes, rmses, r2_scores]
output_data = np.asarray(output_data).transpose().tolist()
print(output_data)
oput = pd.DataFrame(output_data, columns=['L1', 'L2', 'L3', 'AIC', 'MAE',␣
 ↪'RMSE', 'R2'])
# print(oput)
# oput.to_csv('Parametric_space_study.csv', index=False)
print(models[0].summary())
out_path = str(cwd.parent) + '/models/trial0.3.h5'
# models[0].save(out_path)
```

```
[['200', '64', '192', '-42249.02537623041', '0.04866076749821012',
'0.21317433109464726', '0.8131798652629634']]
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 6)]               0

 batch_normalization (BatchN  (None, 6)                24
 ormalization)

 dense (Dense)               (None, 200)               1400

 dropout (Dropout)           (None, 200)               0

 dense_1 (Dense)             (None, 64)                12864

 dropout_1 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 192)               12480

 dropout_2 (Dropout)         (None, 192)               0

 dense_3 (Dense)             (None, 1)                 193

=================================================================
Total params: 26,961
Trainable params: 26,949
Non-trainable params: 12
_________________________________________________________________
None
```

[ ]: