National College of Ireland

Configuration Manual

Supervised Machine-Learning as a Decision Support Aid in Sea Lice Control for Norwegian Salmon Farmers
MSCDATOP

Julia Streckfuss
Student ID: 17158192

School of Computing
National College of Ireland

Supervisor: Dr Catherine Mulwa

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Julia Streckfuss
Student ID:	17158192
Programme:	MSCDATOP
Year:	2023
Module:	Supervised Machine-Learning as a Decision Support Aid in Sea Lice Control for Norwegian Salmon Farmers
Supervisor:	Dr Catherine Mulwa
Submission Due Date:	14/08/2023
Project Title:	Configuration Manual
Word Count:	866
Page Count:	16

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	J. Streckfuss
Date:	17th September 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Julia Streckfuss
17158192

1 Introduction

The following manual provides information on hardware and software configurations as well as code examples to reproduce models and model results as described in the technical report on the conducted study with the title 'Supervised Machine-Learning as a Decision Support Aid in Sea Lice Control for Norwegian Salmon Farmers'.

2 Hardware Configurations

The specifications of the hardware on the local machine used to run the project code are as shown below:

Operating System	Microsoft Windows 10 Home
Version	10.0.19045 Build 19045
OS Manufacturer	Microsoft Corporation
System Name	LAPTOP-4NHM994V
System Manufacturer	HP
System Model	HP 250 G5 Notebook PC
System Type	x64-based PC
System SKU	2EW12ES#ABU
Processor	Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz, 2701 Mhz, 2 Core(s), 4 Logical Processor(s)
BIOS Version/Date	Insyde F.26, 21/04/2017
SMBIOS version	2.8
Embedded Controller Version	62.41
BIOS Mode	UEFI
BaseBoard Manufacturer	HP
BaseBoard Product	81ED
BaseBoard Version	62.41
Platform Role	Mobile
Secure Boot State	On
Windows Directory	C:\WINDOWS
System Directory	C:\WINDOWS\system32
Boot Device	\Device\HarddiskVolume1
Hardware Abstraction Layer	Version = "10.0.19041.2728"
Installed Physical Memory	8.00 GB
Total Physical Memory	7.92 GB
Available Physical Memory	3.17 GB

Total Virtual Memory	14.3 GB
Available Virtual Memory	8.69 GB
Page File Space	6.43 GB
Page File	C:\pagefile.sys
Kernel DMA Protection	Off
Virtual-based security	Running
Virtual-based security available security properties	Base Virtualisation Support, Secure Boot, DMA Protection, Mode Based Execution Control

3 Software Configurations

The following software was used for data extraction and model development:

- Jupyter Notebook (version 6.0.3) using Python (version 3.8.3) for Data Extraction from the BarentsWatch API with Anaconda Prompt for Retrieval of Bearer Authentication Tokens.
- Pycharm (2023.1.2) for Data Pre-Processing, Feature Engineering, Model Fitting and Model Evaluation with Anaconda3 as the underlying Python interpreter (Python 3.8)
- Github Desktop for Version Control.

Required packages were installed through Anaconda Prompt. The following libraries are required to run the project code:

Package	**Version**
curl	7.71.1
requests	2.31.0
numpy	1.21.5
pandas	2.0.2
matplotlib	3.7.1
missingno	0.5.2
scikit-learn	1.2.2
imblearn	0.0
xgboost	1.7.6

4 Data Extraction BarentsWatch API

Data had to be extracted via Bearer Authentication Tokens after free sign-up to the BarentsWatch API service [1] as shown in Figure 2 below:

[1] (BarentsWatch 2023)

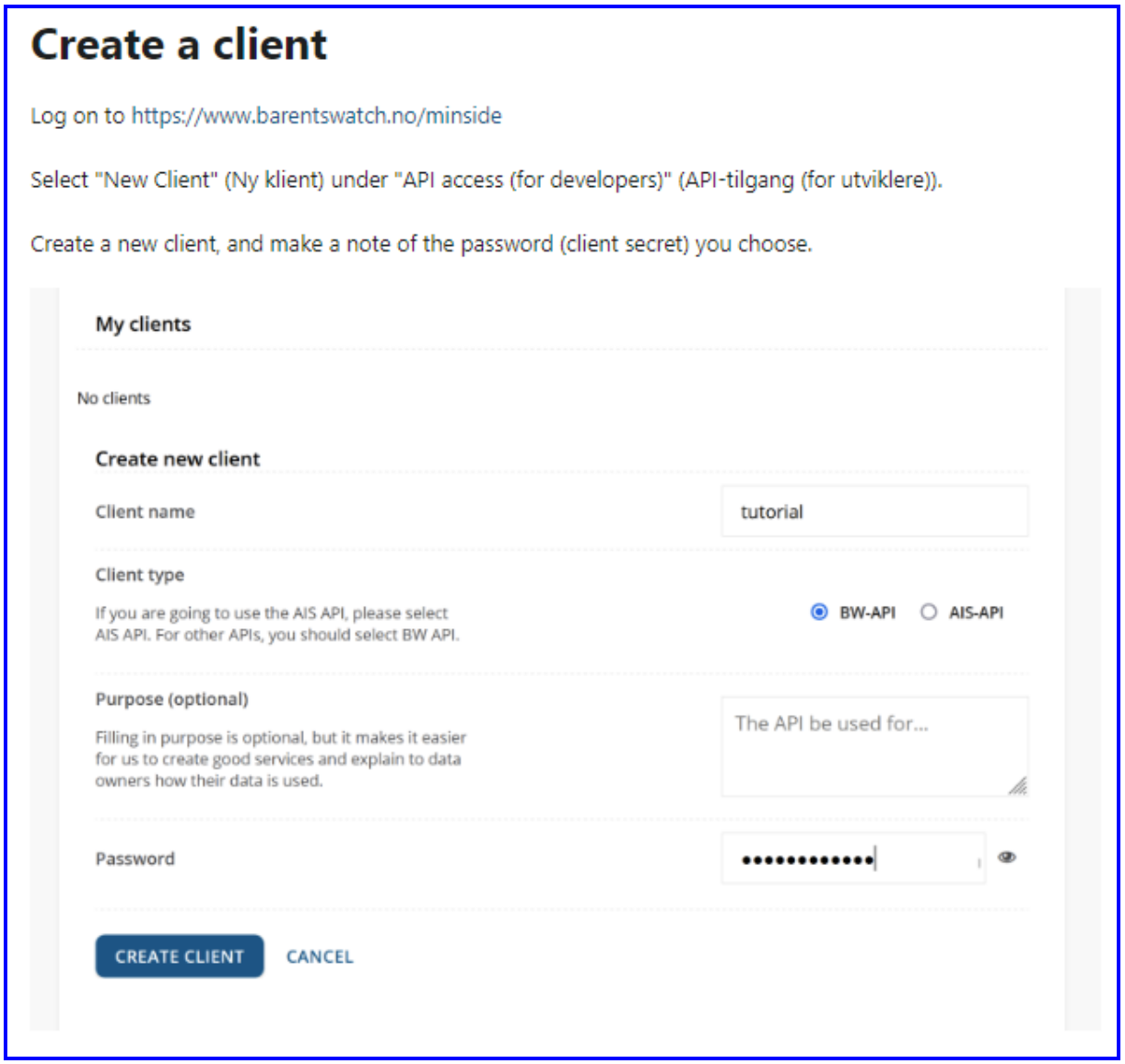

Figure 1: BarentsWatch Tutorial: `https://https://developer.barentswatch.no/docs/tutorial`

Tokens were obtained as demonstrated in Figure 2 by means of the running curl commands in Anaconda Prompt. Access tokens obtained were valid for 1 hour to run requests commands to extract data from the API.

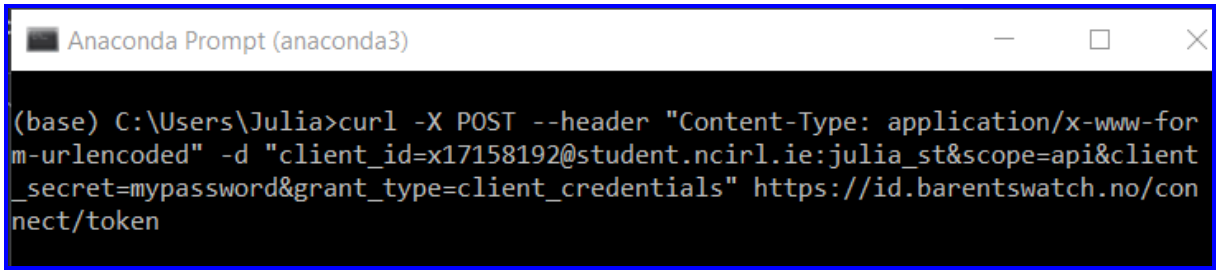

Figure 2: Retrieval Bearer Authentication Token via Curl Command in Anaconda Prompt

```python
# import libraries
import requests
import pandas as pd
import os
```

```python
# fill obtained password through prompt command
password ="password"
```

```python
# create basic API string
url = "https://www.barentswatch.no/bwapi/v1/geodata/fishhealth/locality/"
```

```python
# create Bearer Authentication class: code obtained on website stated below.
# https://stackoverflow.com/questions/29931671/making-an-api-call-in-python-with-an-api-that-requires-a-bearer-token
class BearerClass(requests.auth.AuthBase):
    def __init__(self, token):
        self.token = token
    def __call__(self, r):
        r.headers["authorization"] = "Bearer " + self.token
        return r
```

Figure 3: Preparation for Data Retrieval Functions

After creating the Bearer authentication class and filling the obtained password and creating the base API string as shown in Figure 3, data could be obtained for:

- Weekly fish health per year and week (no specification for farm localities required)
- Detailed farm information per year and week (requires specification of farm locality)
- Lice medication events per year (requires specification of locality)

Efforts were made to reduce number of individual API calls by only extracting detailed farm data and treatments of farms that are not shallow and that have salmonoids.

```python
# function to obtain all weekly fish data for all reported years
def data_collection_all_years(api=url, token=password):
    df = pd.DataFrame()
    localityNo = []
    localityWeekId = []
    name = []
    hasReportedLice = []
    isFallow = []
    avgAdultFemaleLice = []
    hasCleanerfishDeployed = []
    hasMechanicalRemoval = []
    hasSubstanceTreatments = []
    hasPd = []
    hasIla = []
    municipalityNo = []
    municipality = []
    lat =[]
    lon = []
    isOnLand = []
    inFilteredSelection = []
    hasSalmonoids = []
    isSlaughterHoldingCage = []
    week = []
    year = []
    for a in range(2012, 2024):
        for i in range(1,53):
            data = requests.get(f"{api}{a}/{i}",auth=BearerAuth(f"{password}")).json()
            for y in data['localities']:
                localityNo.append(y['localityNo'])
                localityWeekId.append(y['localityWeekId'])
                name.append(y['name'])
                hasReportedLice.append(y['hasReportedLice'])
                isFallow.append(y['isFallow'])
                avgAdultFemaleLice.append(y['avgAdultFemaleLice'])
                hasCleanerfishDeployed.append(y['hasCleanerfishDeployed'])
                hasMechanicalRemoval.append(y['hasMechanicalRemoval'])
                hasSubstanceTreatments.append(y['hasSubstanceTreatments'])
                hasPd.append(y['hasPd'])
                hasIla.append(y['hasIla'])
                municipalityNo.append(y['municipalityNo'])
                municipality.append(y['municipality'])
                lat.append(y['lat'])
                lon.append(y['lon'])
                isOnLand.append(y['isOnLand'])
                inFilteredSelection.append(y['inFilteredSelection'])
                hasSalmonoids.append(y['hasSalmonoids'])
                isSlaughterHoldingCage.append(y['isSlaughterHoldingCage'])
                week.append(i)
                year.append(a)
    df['localityNo'] = localityNo
    df['localityWeekId'] = localityWeekId
    df['name'] = name
    df['hasReportedLice'] = hasReportedLice
    df['isFallow'] = isFallow
    df['avgAdultFemaleLice'] = avgAdultFemaleLice
    df['hasCleanerfishDeployed'] = hasCleanerfishDeployed
    df['hasMechanicalRemoval'] = hasMechanicalRemoval
    df['hasSubstanceTreatments'] = hasSubstanceTreatments
    df['hasPd'] = hasPd
    df['hasIla'] = hasIla
    df['municipalityNo'] = municipalityNo
    df['municipality'] = municipality
    df['lat'] = lat
    df['lon'] = lon
    df['isOnLand'] = isOnLand
    df['inFilteredSelection'] = inFilteredSelection
    df['hasSalmonoids'] = hasSalmonoids
    df['isSlaughterHoldingCage'] = isSlaughterHoldingCage
    df['week'] = week
    df['year'] = year
    os.chdir('C:/Users/Julia/Downloads')
    df.to_csv("fish_data_all_years.csv")
    return df
```

Figure 4: Weekly Fish Data Extraction and Local Storage

```python
# function to obtain detailed farm data for all weeks in a given year
# for years with 53 years, range needs to changed to (1,54)
def site_details_week(year, localityno, api=url, token=password):
    df = pd.DataFrame()
    years = []
    week = []
    localityNo = []
    avgMobileLice = []
    avgStationaryLice = []
    seaTemperature = []
    timeSinceLastChitinSynthesisInhibitorTreatment = []
    placement = []
    capacity = []
    unit = []
    purposes = []
    productionTypes = []
    species = []
    speciesList = []
    isGreen = []
    localityIsInAquaCultureRegister = []


    for a in range(1,53):
        data = requests.get(f"{api}{localityno}/{year}/{a}",auth=BearerAuth(f"{password}")).json()
        years.append(year)
        week.append(a)
        localityNo.append(localityno)
        avgMobileLice.append(data['localityWeek']['avgMobileLice'])
        avgStationaryLice.append(data['localityWeek']['avgStationaryLice'])
        seaTemperature.append(data['localityWeek']['seaTemperature'])
        timeSinceLastChitinSynthesisInhibitorTreatment.append(data['localityWeek']['timeSinceLastChitinSynthesisInhibitorTreatme
        placement.append(data['aquaCultureRegister']['placement'])
        capacity.append(data['aquaCultureRegister']['capacity'])
        unit.append(data['aquaCultureRegister']['unit'])
        purposes.append(data['aquaCultureRegister']['purposes'])
        productionTypes.append(data['aquaCultureRegister']['productionTypes'])
        species.append(data['aquaCultureRegister']['species'])
        speciesList.append(data['aquaCultureRegister']['speciesList'])
        isGreen.append(data['aquaCultureRegister']['isGreen'])
        localityIsInAquaCultureRegister.append(data['localityIsInAquaCultureRegister'])

    df['year'] = years
    df['week'] = week
    df['localityNo'] = localityNo
    df['avgMobileLice'] = avgMobileLice
    df['avgStationaryLice'] = avgStationaryLice
    df['seaTemperature'] = seaTemperature
    df['timeSinceLastChitinSynthesisInhibitorTreatment'] = timeSinceLastChitinSynthesisInhibitorTreatment
    df['placement'] = placement
    df['capacity'] = capacity
    df['unit'] = unit
    df['purposes'] = purposes
    df['productionTypes'] = productionTypes
    df['species'] = species
    df['speciesList'] = speciesList
    df['isGreen'] = isGreen
    df['localityIsInAquaCultureRegister'] = localityIsInAquaCultureRegister


    os.chdir(f"C:/Users/Julia/Desktop/Masters/Data/detailed/{year}")
    df.to_csv(f"data_detailed_{year}_{localityno}.csv")
    return df
```

Figure 5: Detailed Farm Data Extraction and Local Storage

```python
# append all files of one year to one csv file and store locally
def append_all_years(path, name):
    files = os.listdir(path)
    combined = pd.DataFrame()
    os.chdir(path)
    for i in files:
        df = pd.read_csv(i)
        combined = combined.append(df, ignore_index=True)

    os.chdir('C:/Users/Julia/Desktop/Masters/Data/detailed/all_years')
    combined.to_csv(f"{name}_all_farms.csv")
    return combined
```

```python
# append all years' files to one csv file and store locally
def append_all_years2(path, name):
    files = os.listdir(path)
    combined = pd.DataFrame()
    os.chdir(path)
    for i in files:
        df = pd.read_csv(i)
        combined = combined.append(df, ignore_index=True)

    os.chdir('C:/Users/Julia/Desktop/Masters/Data/detailed/all_years_combined')
    combined.to_csv(f"{name}_all_farms.csv")
    return combined
```

Figure 6: Combine Data of All Reported Years

```python
# obtain fish data all years: result from fish_data_collection_weekly_fish_health_data
all_years = pd.read_csv('fish_data_all_years.csv')
```

```python
all_years.week.unique()
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
       52, 53], dtype=int64)
```

```python
# check for only active salmon farms data
all_years_salmon_production = all_years.loc[(all_years.isFallow==False)&(all_years.hasSalmonoids==True)].copy()
```

```python
# check for only those farms that have mechanical removal
mech_remov = all_years_salmon_production.loc[all_years_salmon_production.hasMechanicalRemoval==True].copy()
```

```python
# check for only those farms that have cleaner fish deployed
cleaner_fish = all_years_salmon_production.loc[all_years_salmon_production.hasCleanerfishDeployed==True].copy()
```

```python
# check for only those farms that have substance treatments applied
substance_df = all_years_salmon_production.loc[all_years_salmon_production.hasSubstanceTreatments==True].copy()
```

```python
# obtain mechanical removals data
def mechanicalremovals(year, api=url, token=password):
    df = pd.DataFrame()
    years = []
    week = []
    localityNo = []
    mechanicalRemoval = []
    mechanicalRemovalEntireLocality = []
    mech_remov_year = mech_remov.loc[mech_remov.year == year].copy()
    localitylist = mech_remov_year['localityNo'].unique().tolist()

    for i in localitylist:
        data = requests.get(f"{api}{i}/liceMedicationEvents/{year}",auth=BearerAuth(f"{password}")).json()
        data = data['data']
        for y in data:
            if y['mechanicalRemoval'] == True:
                mechanicalRemoval.append(y['mechanicalRemoval'])
                mechanicalRemovalEntireLocality.append(y['mechanicalRemovalEntireLocality'])
                years.append(year)
                week.append(y['week'])
                localityNo.append(i)
    df['year'] = years
    df['week'] = week
    df['localityNo'] = localityNo
    df['mechanicalRemoval'] = mechanicalRemoval
    df['mechanicalRemovalentirelocality'] = mechanicalRemovalEntireLocality
    os.chdir('C:/Users/Julia/Desktop/Masters/Data/treatments/mech_removal')
    df.to_csv(f"fish_data_mechanical_removals_{year}.csv")
    return df
```

Figure 7: Extraction Treatment Data: Example Mechanical Removal Data

5 Data Integration and Cleaning

Numerous steps were conducted to integrate weekly fish health data, detailed weekly farm data, and treatment data, and to clean the data and interpolate missing values. Coding was run in Pycharm. See detailed code in py file provided as part of the code artifact:

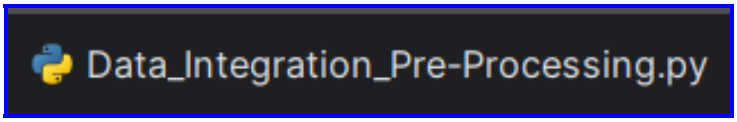

Figure 8: Data Integration and Pre-Processing Python script

6 Feature Engineering

Feature Engineering was conducted as part of Figure 8 and detailed code can be found there. One example of the code is shown in Figure 9 below. It shows how lice figure and sea temeperature aggregates were obtained from farms in 20km haversine distance to a respective farm locality.

```
# get localities in 20 km distance and derive lice counts and sea temp
dist = DistanceMetric.get_metric('haversine')
dates = []
years = []
weeks = []
localitynos = []
female = []
female_max = []
mobile = []
mobile_max = []
stationary = []
stationary_max = []
temp = []
temp_max = []

for i in interpolated_df_clean8.date.unique():
    week_data = interpolated_df_clean8.loc[interpolated_df_clean8.date== i].copy()
    year = week_data.year.unique()
    year_value = year[0]
    week = week_data.week.unique()
    week_value = week[0]
    week_data['lat_coord'] = np.radians(week_data['lat_coord'])
    week_data['lon_coord'] = np.radians(week_data['lon_coord'])
    df = pd.DataFrame(dist.pairwise(week_data[['lat_coord','lon_coord']].to_numpy())*6371,  columns=week_data.localityNo.unique(), index=week_data.localityNo.unique())
    week_data.set_index('localityNo',inplace=True)
    for y in df.columns:
        farms_within_distance = df.index[(df[y] <= 20) &(df[y] > 0)].to_list()
        female_distance = week_data.loc[week_data.index.isin(farms_within_distance)]['av_ad_female'].mean()
        female_distance_max = week_data.loc[week_data.index.isin(farms_within_distance)]['av_ad_female'].max()
        mobile_distance = week_data.loc[week_data.index.isin(farms_within_distance)]['av_mobile'].mean()
        mobile_distance_max = week_data.loc[week_data.index.isin(farms_within_distance)]['av_mobile'].max()
        stationary_distance = week_data.loc[week_data.index.isin(farms_within_distance)]['av_stationary'].mean()
        stationary_distance_max = week_data.loc[week_data.index.isin(farms_within_distance)]['av_stationary'].max()
        temp_distance = week_data.loc[week_data.index.isin(farms_within_distance)]['sea_temp'].mean()
        temp_distance_max = week_data.loc[week_data.index.isin(farms_within_distance)]['sea_temp'].max()
        dates.append(i)
        years.append(year_value)
        weeks.append(week_value)
        localitynos.append(y)
        female.append(female_distance)
        female_max.append(female_distance_max)
        mobile.append(mobile_distance)
        mobile_max.append(mobile_distance_max)
        stationary.append(stationary_distance)
        stationary_max.append(stationary_distance_max)
        temp.append(temp_distance)
        temp_max.append(temp_distance_max)
```

Figure 9: Lice and Sea Temperature Aggregates Retrieval of Farms in 20km Distance

7 Data Preparation for Modelling

- Common techniques applied to data before model fitting were:
- Dummy Variable Creation for Categorical Variables.
- Scaling of Numerical Features for KNN and XGBoost algorithm.
- Oversampling of Minority Class via SMOTE package.
- Split of Training and Test set.

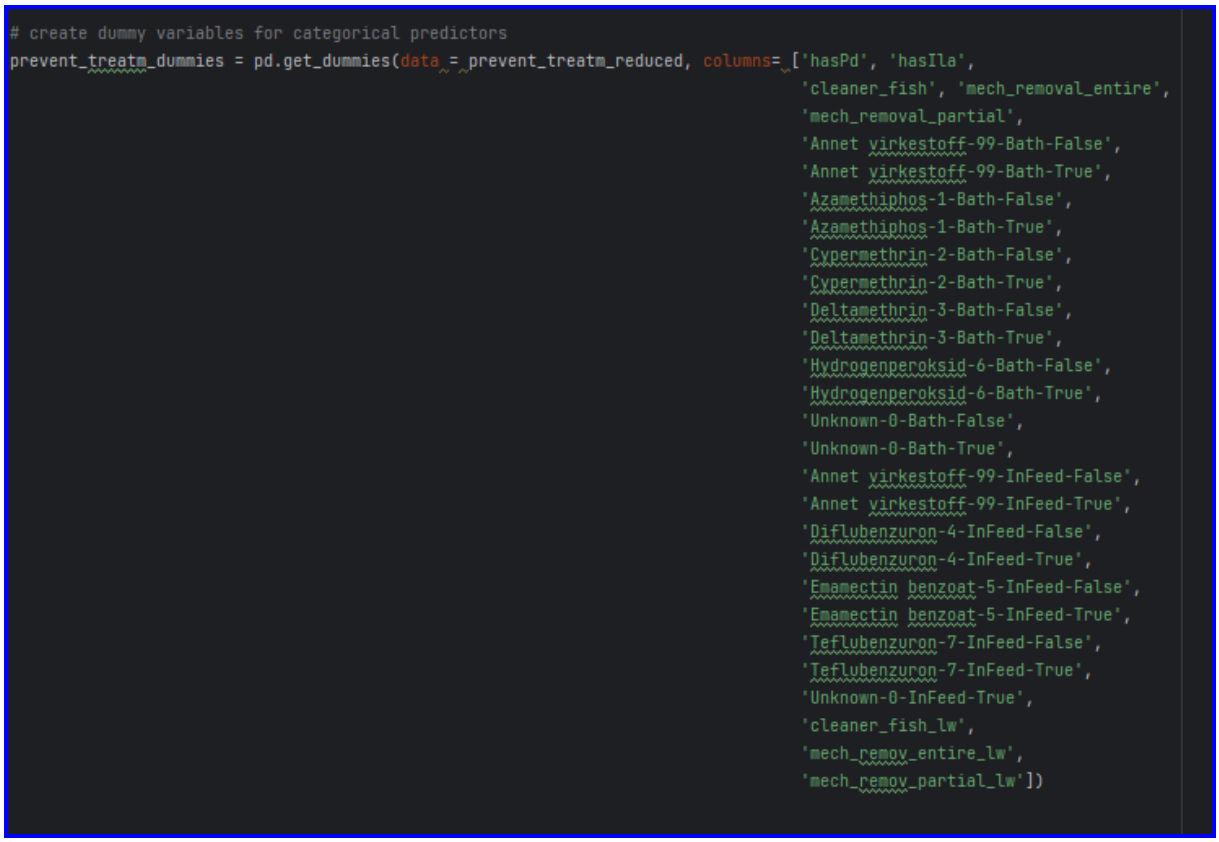

Figure 10: Dummy Variable Creation

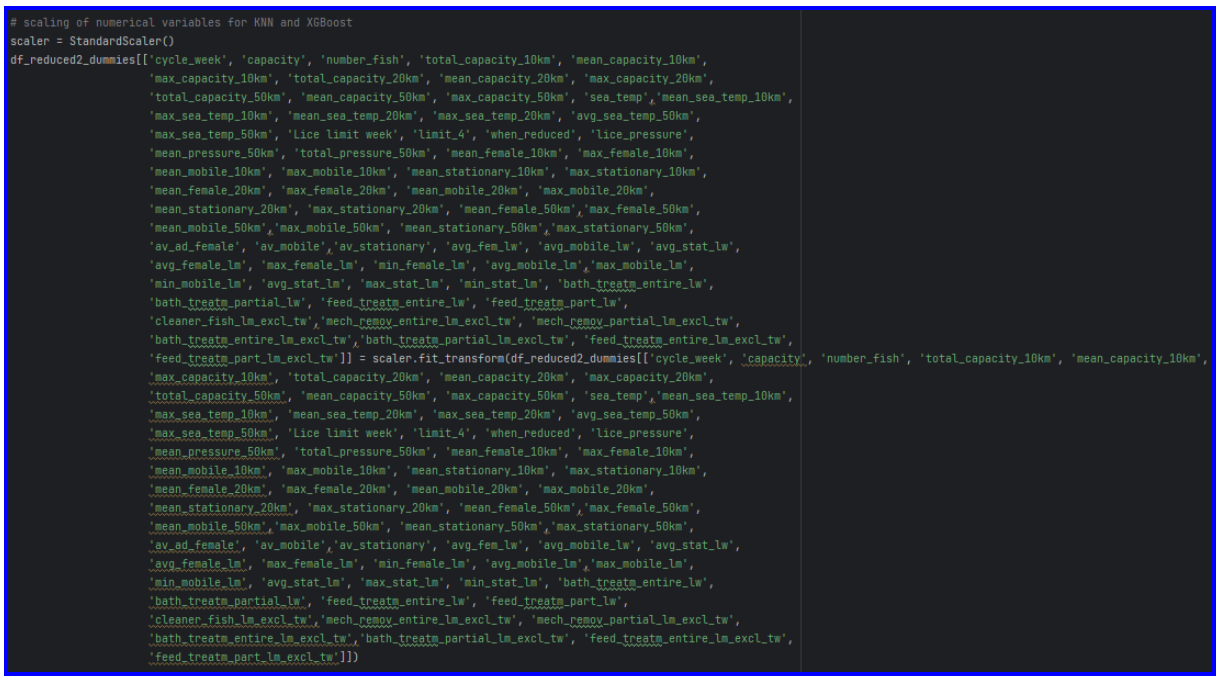

Figure 11: Scaling of Numerical Variables for KNN and XGBoost

```
# oversampling of minority class of farms that exceed threshold despite preventative treatment
SEED = 1
smote = SMOTE(random_state = SEED)
X, y = smote.fit_resample(prevent_treatm_dummies[['cycle_week',  'capacity', 'number_fish', 'sea_temp',
                                                  'avg_sea_temp_50km', 'max_sea_temp_50km', 'Lice limit week',
                                                  'lice_pressure', 'mean_pressure_50km', 'total_pressure_50km',
                                                  'mean_female_50km', 'mean_mobile_50km', 'mean_stationary_50km',
                                                  'av_ad_female', 'av_mobile', 'av_stationary', 'avg_fem_lw',
                                                  'avg_mobile_lw', 'avg_stat_lw', 'avg_female_lm', 'max_female_lm',
                                                  'min_female_lm', 'avg_mobile_lm', 'max_mobile_lm', 'min_mobile_lm',
                                                  'avg_stat_lm', 'max_stat_lm', 'min_stat_lm',
                                                  'number_substance_treatments', 'bath_treatments_entire_locality',
                                                  'bath_treatments_partial_locality', 'feed_treatments_entire_locality',
                                                  'feed_treatments_partial_locality', 'bath_treatm_entire_lw',
                                                  'bath_treatm_partial_lw', 'feed_treatm_entire_lw',
                                                  'feed_treatm_part_lw', 'cleaner_fish_lm_excl_tw',
                                                  'mech_remov_entire_lm_excl_tw', 'mech_remov_partial_lm_excl_tw',
                                                  'bath_treatm_entire_lm_excl_tw', 'bath_treatm_partial_lm_excl_tw',
                                                  'feed_treatm_entire_lm_excl_tw', 'feed_treatm_part_lm_excl_tw',
                                                  'hasPd_0', 'hasPd_1', 'hasIla_0', 'hasIla_1', 'cleaner_fish_0',
                                                  'cleaner_fish_1', 'mech_removal_entire_0', 'mech_removal_entire_1',
                                                  'mech_removal_partial_0', 'mech_removal_partial_1',
                                                  'Annet virkestoff-99-Bath-False_0',
                                                  'Annet virkestoff-99-Bath-False_1', 'Annet virkestoff-99-Bath-True_0',
                                                  'Annet virkestoff-99-Bath-True_1', 'Azamethiphos-1-Bath-False_0',
                                                  'Azamethiphos-1-Bath-False_1', 'Azamethiphos-1-Bath-True_0',
                                                  'Azamethiphos-1-Bath-True_1', 'Cypermethrin-2-Bath-False_0',
                                                  'Cypermethrin-2-Bath-False_1', 'Cypermethrin-2-Bath-True_0',
                                                  'Cypermethrin-2-Bath-True_1', 'Deltamethrin-3-Bath-False_0',
                                                  'Deltamethrin-3-Bath-False_1', 'Deltamethrin-3-Bath-True_0',
                                                  'Deltamethrin-3-Bath-True_1', 'Hydrogenperoksid-6-Bath-False_0',
                                                  'Hydrogenperoksid-6-Bath-False_1', 'Hydrogenperoksid-6-Bath-True_0',
                                                  'Hydrogenperoksid-6-Bath-True_1', 'Unknown-0-Bath-False_0',
                                                  'Unknown-0-Bath-False_1', 'Unknown-0-Bath-True_0',
                                                  'Unknown-0-Bath-True_1', 'Annet virkestoff-99-InFeed-False_0',
                                                  'Annet virkestoff-99-InFeed-False_1',
                                                  'Annet virkestoff-99-InFeed-True_0',
                                                  'Annet virkestoff-99-InFeed-True_1', 'Diflubenzuron-4-InFeed-False_0',
                                                  'Diflubenzuron-4-InFeed-False_1', 'Diflubenzuron-4-InFeed-True_0',
                                                  'Diflubenzuron-4-InFeed-True_1', 'Emamectin benzoat-5-InFeed-False_0',
                                                  'Emamectin benzoat-5-InFeed-False_1', 'Emamectin benzoat-5-InFeed-True_0',
                                                  'Emamectin benzoat-5-InFeed-True_1', 'Teflubenzuron-7-InFeed-False_0',
                                                  'Teflubenzuron-7-InFeed-False_1', 'Teflubenzuron-7-InFeed-True_0',
                                                  'Teflubenzuron-7-InFeed-True_1',
                                                  'Unknown-0-InFeed-True_0', 'Unknown-0-InFeed-True_1',
                                                  'cleaner_fish_lw_0', 'cleaner_fish_lw_1', 'mech_remov_entire_lw_0',
                                                  'mech_remov_entire_lw_1', 'mech_remov_partial_lw_0',
                                                  'mech_remov_partial_lw_1']], prevent_treatm_dummies['never_over_limit'])
```

Figure 12: Oversampling with SMOTE

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2, stratify= y, random_state=SEED)
```

Figure 13: Split of Training and Test Set

8 Pre-Modeling Analysis

For pre-modelling, a pruned CART was fit by means of grid search and 10-fold cross-validation and cues obtained from tree visual were used as a filtering criteria to obtain mean number of weeks till lice threshold exceedance when cues under given conditions are not acted upon in the form of preventative treatment. See code in visual below:

```
# Grid Search for best parameters
params_dt = {'max_depth':[3,4,5,6,7,8,9,10],
             'min_samples_leaf':[0.04,0.06,0.08,0.1],
             'max_features':[0.2,0.4,0.6,0.8]}
grid_dt = GridSearchCV(estimator=dt,
                       param_grid=params_dt,
                       scoring='accuracy',
                       cv=10,
                       n_jobs=-1)
grid_dt.fit(X_train,y_train)
best_hyperparams = grid_dt.best_params_
print('Best set of hyperparameters:\n', best_hyperparams)
best_CV_score = grid_dt.best_score_
print('Best obtained CV accuracy:\n',best_CV_score)
best_model = grid_dt.best_estimator_
print('Best performing model:\n', best_model)
test_acc = best_model.score(X_test, y_test)
print('Test-set accuracy of best performing model:\n', test_acc)
print(classification_report(y_test,best_model.predict(X_test)))

y_pred_proba = best_model.predict_proba(X_test)[:,1]
rf_roc_auc_score = roc_auc_score(y_test, y_pred_proba)
print(rf_roc_auc_score)
RocCurveDisplay.from_predictions(y_test,y_pred_proba)

# tree visual preparation
X = prevent_treat_even.drop(['never_over_limit'],axis=1)
y = prevent_treat_even['never_over_limit'].copy()
dt_feature_names = list(X.columns)
dt_target_names = [str(s) for s in y.unique()]

fig = plt.figure(figsize=(20,15))
tree.plot_tree(best_model, feature_names = dt_feature_names, class_names = dt_target_names, label='all',filled=True, fontsize=8)


# under_threshold when cues were missed: max_female_lm > 0.16 and sea_temp >7.521 and av_stat_lm > 0.0 and no treatment
missed_cues = under_threshold.loc[(under_threshold['max_female_lm']>0.16)&(under_threshold['avg_sea_temp_50km']>7.521) &(under_threshold['min_stat_lm']>0)&(under_threshold['treatment']==0)].copy()
missed_cues['when_over_limit'].value_counts()
missed_cues.describe()

# remove observations that never went over
missed_cues2 = missed_cues.loc[missed_cues.when_over_limit<9].copy()

# assess mean and median number of weeks until exceedance to obtain best point of warning
missed_cues2.when_over_limit.describe()
missed_cues2.when_over_limit.value_counts()

plt.hist(missed_cues2['when_over_limit'], color = 'blue', edgecolor = 'black', bins=8)
```

Figure 14: Missed Treatment Cues

9 Model Fitting and Evaluation Exceedance Classifiers

Supervised classification models were fit to obtain general risk of exceedance for 4, 6, and 8 weeks ahead. Code was the same for all 3 time frames and fitting and evaluation code is shown below for all models except CART, because CART was applied in the same manner as in pre-modelling analysis.

```
# Random Forest
rf = RandomForestClassifier(n_estimators=300, random_state=SEED)
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
rf_accuracy = accuracy_score(y_test, y_pred)
print('Accuracy of Random Forest Classifier:{:.3f}'.format(rf_accuracy))
print(classification_report(y_test,rf.predict(X_test)))
rf.get_params()


cm_rf = confusion_matrix(y_test,y_pred)
disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf)
disp_rf.plot()
plt.show()
y_pred_proba = rf.predict_proba(X_test)[:,1]
rf_roc_auc_score = roc_auc_score(y_test, y_pred_proba)
print(rf_roc_auc_score)
RocCurveDisplay.from_predictions(y_test,y_pred_proba)
```

Figure 15: Random Forest Classifier: Treatment Current Week not Included as Predictor

```
# Adaboost Classifier
stump = DecisionTreeClassifier(max_depth=1, random_state=SEED)
ada_cl = AdaBoostClassifier(base_estimator=stump, n_estimators=300)
ada_cl.fit(X_train, y_train)
y_pred = ada_cl.predict(X_test)
ada_cl_accuracy = accuracy_score(y_test, y_pred)
print('Accuracy of AdaBoost Classifier:{:.3f}'.format(ada_cl_accuracy))
print(classification_report(y_test,ada_cl.predict(X_test)))

y_pred_proba = ada_cl.predict_proba(X_test)[:,1]
ada_cl_roc_auc_score = roc_auc_score(y_test, y_pred_proba)
print(ada_cl_roc_auc_score)
RocCurveDisplay.from_predictions(y_test,y_pred_proba)

cm_ada = confusion_matrix(y_test,y_pred)
cm_ada = ConfusionMatrixDisplay(confusion_matrix=cm_ada)
cm_ada.plot()
plt.show()
# GradientBoosting Classifier
gbt_cl = GradientBoostingClassifier(n_estimators=300, max_depth=1, random_state=SEED)
gbt_cl.fit(X_train, y_train)
y_pred = gbt_cl.predict(X_test)
gbt_cl_accuracy = accuracy_score(y_test, y_pred)
print('Accuracy of GradientBoosting Classifier:{:.3f}'.format(gbt_cl_accuracy))
print(classification_report(y_test,gbt_cl.predict(X_test)))
y_pred_proba = gbt_cl.predict_proba(X_test)[:,1]
gbt_cl_roc_auc_score = roc_auc_score(y_test, y_pred_proba)
print(gbt_cl_roc_auc_score)
RocCurveDisplay.from_predictions(y_test,y_pred_proba)
```

Figure 16: AdaBoost and Gradient Boosting Classifier: Treatment Current Week not Included as Predictor

```
# KNN
knn_cl = KNeighborsClassifier(n_neighbors=604)
knn_cl.fit(X_train, y_train)
y_pred = knn_cl.predict(X_test)
knn_cl_test_accuracy = accuracy_score(y_test,y_pred)
print('Accuracy of KNN Classifier: {:.3f}'.format(knn_cl_test_accuracy))
print(classification_report(y_test,knn_cl.predict(X_test)))
y_pred_proba = knn_cl.predict_proba(X_test)[:,1]
knn_cl_roc_auc_score = roc_auc_score(y_test, y_pred_proba)
print(knn_cl_roc_auc_score)
RocCurveDisplay.from_predictions(y_test,y_pred_proba)
```

Figure 17: KNN Classifier: Treatment Current Week not Included as Predictor

```
# XGBOOST
xgb_cl = xgb.XGBClassifier(random_state=SEED)
xgb_cl.fit(X_train, y_train)
y_pred = xgb_cl.predict(X_test)
xgb_cl_test_accuracy = accuracy_score(y_test,y_pred)
print('Accuracy of XGBoost Classifier: {:.3f}'.format(xgb_cl_test_accuracy))
print(classification_report(y_test,xgb_cl.predict(X_test)))
y_pred_proba = xgb_cl.predict_proba(X_test)[:,1]
xgb_roc_auc_score = roc_auc_score(y_test, y_pred_proba)
print(xgb_roc_auc_score)
RocCurveDisplay.from_predictions(y_test,y_pred_proba)
cm_xg = confusion_matrix(y_test,y_pred)
cm_xg = ConfusionMatrixDisplay(confusion_matrix=cm_xg)
cm_xg.plot()
plt.show()
```

Figure 18: XGBoost Classifier: Treatment Current Week not Included as Predictor

10 Model Fitting Random Forest and Treatment Recommendations

As a last step, predicted exceeders from RF fit without treatment in current week were fed into Random Forest trained including treatment of current week. Predictions were made for no treatment applied versus all other possible treatment combinations and

risk reduction was displayed for those farm localities that by means of applying specfic treatment can turn the prediction to exceed to the prediction not exceed. This was only conducted for exceedance 4 weeks after current week. See code below.

```
# predict for all possible treatments
exceed_pred_exploded_dummies_pred = exceed_pred_exploded_dummies.drop(columns=['real_outcome','pred','false_preds','prediction_prob','ident'],axis=1)
treatment_pred = rf_treatm.predict(exceed_pred_exploded_dummies_pred)
treatment_pred_proba = rf_treatm.predict_proba(exceed_pred_exploded_dummies_pred)[:,1]
exceed_pred_exploded_dummies['treatment_pred'] = treatment_pred
exceed_pred_exploded_dummies['treatment_pred_prob'] = treatment_pred_proba
exceed_pred_exploded_dummies['treatm_num'] = treatm_id
print(exceed_pred_exploded_dummies.ident.dtypes)

# baseline treatment is no treatment applied
baseline = exceed_pred_exploded_dummies.loc[exceed_pred_exploded_dummies.treatm_num==188].copy()
baseline_vars = baseline[['ident', 'treatment_pred_prob']].copy()
baseline_vars.drop_duplicates(keep='first',inplace=True)
baseline_vars = baseline_vars.rename(columns={'treatment_pred_prob':'baseline_risk'})
exceed_pred_exploded_dummies = exceed_pred_exploded_dummies.merge(baseline_vars, on=['ident'])

# create risk reduction from baseline risk minus treatment outcome probability
exceed_pred_exploded_dummies['risk_reduction'] = exceed_pred_exploded_dummies['baseline_risk'] - exceed_pred_exploded_dummies['treatment_pred_prob']
exceed_pred_exploded_dummies.rename(columns={'prediction_prob':'prob_to_exceed', 'treatment_pred_prob':'prob_to_exceed_with_treatment'},inplace=True)

# obtain treatment description
reduced_treatm_desc = treatment_combis[['treatm_num','treatm_combination']].copy()

exceed_pred_exploded_dummies = exceed_pred_exploded_dummies.merge(reduced_treatm_desc, how='left', on='treatm_num')
exceed_pred_exploded_dummies['farm_week_ident'] = farm_week_id
exceed_pred_exploded_dummies.rename(columns={'treatm_combination':'proposed_treatment'},inplace=True)

# create recommendations for farms where treatment prediction is to not exceed and where probability lies under 0.5
avoid_exceed = exceed_pred_exploded_dummies.loc[(exceed_pred_exploded_dummies.treatment_pred==0)&(exceed_pred_exploded_dummies.prob_to_exceed_with_treatment<0.5)].copy()

# create recommendations for farmers by displaying the most efficient treatments:
# those that show lowest probability to exceed
for i in avoid_exceed['farm_week_ident'].unique():
    data = avoid_exceed.loc[avoid_exceed['farm_week_ident']==i].copy()
    data.sort_values(by=['prob_to_exceed_with_treatment'],ascending=True,inplace=True)
    prob_exceed = data['prob_to_exceed'].iloc[0]
    data_reduced= data[['proposed_treatment','baseline_risk','prob_to_exceed_with_treatment','risk_reduction']].copy()
    # create printed recommendation for farmers
    print(f"Predicted risk for lice threshold exceedance in 4 weeks is at {prob_exceed:.2f}. \n See predicted risk reduction over baseline risk of no treatment by applying following proposed treatments:\n")
    print(data_reduced.iloc[:3].to_string(index=False))
    print("\n")
```

Figure 19: Probability Reduction to Exceed in 4 Weeks Time including 3 most Efficient Treatment Recommendations

References

BarentsWatch (2023), `https://https://developer.barentswatch.no/docs/tutorial`. Accessed: 2023-04-15.