# Golden redfish in Icelandic waters: a statistical approach

# Software and data

## Packages imported

This file contains the python code and description of the Golden redfish statistical model that Maris Optimum has developed.

Sources:

Data from Icelandic Marine and Freshwater Institute (IMFI)

Data from the Directory of Fisheries (DF) obtained from IMFI

Gradient boosting regression algorithm xgboost and various packages

Python 3.11 programming language

The main packages used are: [the xgboost regressor](https://xgboost.readthedocs.io/en/stable/index.html), [the scikit-learn suite](https://scikit-learn.org/stable/user_guide.html), [the shap suite](https://shap.readthedocs.io/en/latest/index.html).

We start by describing the data gathering and the functions of the model.

"""Optimal regression found to estimate catch sizes."""  
  
  
import pandas as pd  
import math  
import seaborn as sns  
from matplotlib import pyplot as plt  
# from sklearn.preprocessing import StandardScaler  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import mean\_absolute\_error  
from sklearn.metrics import mean\_squared\_error  
from sklearn.metrics import explained\_variance\_score  
from sklearn.metrics import r2\_score  
from sklearn.linear\_model import LinearRegression  
from sklearn.preprocessing import StandardScaler  
from sklearn.decomposition import PCA  
import xgboost as xgb  
from sklearn.model\_selection import GridSearchCV  
import shap  
import time  
  
path\_str = 'R:\\Ráðgjöf\\Maris Optimum/gr\_data\\'  
# path\_str = ''  
path\_grs\_weight =\  
 'R:/Ráðgjöf/Maris Optimum/Data\_from\_Hafro\_20220906/ssb\_weights/'  
  
red\_length\_weights = pd.read\_csv(  
 path\_grs\_weight + "red\_length\_weights.csv")  
red\_length\_weights.set\_index('ar', inplace=True)

## Data Fetched and adjusted

### Data

def get\_new\_data(fractile, show\_heatmaps):  
 """  
 Fetch all data for the regression.  
  
 Parameters  
 ----------  
 fractile : integer.  
  
 Returns  
 -------  
 XX\_df : Dataframe with data for all the dependent variables.  
 YY : Dataframe with data for the the independent variable.  
  
 """  
 X100\_df = pd.read\_csv(path\_str + 'distribution100.csv',  
 sep=",")  
 ysq\_df = X100\_df[['ar', 'max(cum)']].copy()  
 ysq\_df.set\_index(['ar'], inplace=True)  
 ysq\_df = ysq\_df[~ysq\_df.index.duplicated(keep='first')]  
  
 X\_df = pd.read\_csv(path\_str + 'distribution' + fractile + '.csv',  
 sep=",")  
  
 catch\_df = pd.read\_csv(path\_str + 'golden\_redfish\_catch.csv',  
 sep=";")  
  
 catch\_df.at[37, 'year'] = 2022.0  
 catch\_df.at[37, 'catch'] = 26  
 catch\_df.at[37, 'number'] = 29  
  
 X\_cal\_df = pd.read\_csv(path\_str + 'distribution\_commercial.csv',  
 sep=",")  
 X\_cal\_df.drop(1605, axis=0, inplace=True)  
  
 X\_cal\_df = X\_cal\_df.pivot(index='ar',  
 columns='lengd',  
 values='per\_length')  
 X\_cal\_df = X\_cal\_df.fillna(0)  
 X\_cal\_df.columns = 1000 + X\_cal\_df.columns  
 X\_cal\_df.columns = X\_cal\_df.columns.astype(int).astype(str)  
  
 catch\_df = catch\_converter(X\_cal\_df, catch\_df, show\_heatmaps)  
 catch\_df.year = catch\_df.year.astype(int)  
 catch\_df.set\_index(catch\_df.year, inplace=True)  
  
 X\_cal\_df = X\_cal\_df.mul(catch\_df.number\*-1e6, axis=0)  
  
 XX\_df = X\_df.pivot(index='ar',  
 columns='lengd',  
 values='per\_length')  
  
 XX\_df = pd.merge(XX\_df, ysq\_df, right\_index=True, left\_index=True)  
  
 XX\_df.drop(11.9, axis=1, inplace=True)  
 XX\_df.drop(12.5, axis=1, inplace=True)  
 XX\_df.drop(12.6, axis=1, inplace=True)  
 XX\_df.drop(13.1, axis=1, inplace=True)  
 XX\_df.drop(13.4, axis=1, inplace=True)  
 XX\_df.drop(13.6, axis=1, inplace=True)  
 XX\_df.drop(13.7, axis=1, inplace=True)  
 XX\_df.drop(13.9, axis=1, inplace=True)  
 XX\_df.drop(14.4, axis=1, inplace=True)  
 XX\_df.drop(14.5, axis=1, inplace=True)  
 XX\_df.drop(14.7, axis=1, inplace=True)  
 XX\_df.drop(14.8, axis=1, inplace=True)  
 XX\_df.drop(14.9, axis=1, inplace=True)  
  
 XX\_df.columns = XX\_df.columns.astype(str)  
  
 YX = pd.read\_csv(path\_str+"RED\_numbers\_at\_age.csv", sep=";")  
 YY = YX.iloc[15:53, 28]  
  
 XX\_df = XX\_df.join(X\_cal\_df.iloc[:, :])  
  
 XX\_df.index = XX\_df.index.astype(str)  
  
 s = XX\_df.index[27:35]  
 s.index = XX\_df[27:35]  
 s = pd.get\_dummies(s)  
 s.index = XX\_df.index[27:35]  
 XX\_df = XX\_df.join(s)  
  
 XX\_df = XX\_df.fillna(0)  
 if show\_heatmaps:  
 heat\_maps(XX\_df,YY)  
 return (XX\_df, YY)

### Conversion of kg catch to units of individuals in length groups

The annual catch of the fleet is reported by weight. The statistical model is based on number of individuals at each length cm. Measurement data from IMFI surveys was employed to estmate the catch into lengths and number of individuals.

Where data did not exist polynomial regression of the second order was employed to estmiate lengths and weights.

def catch\_converter(X\_catch\_per\_df, catch\_df, show\_heatmaps):  
 """  
 Catch information used to formulate and calculate units from kg.  
  
 Parameters  
 ----------  
 X\_catch\_per\_df : Dataframe containing percentages of catch.  
 catch\_df : Dataframe containing total cath in kg.  
  
 Returns  
 -------  
 catch\_df : Dataframe containing total catch in lengths.  
 """  
 path\_grs\_str = 'R:/Ráðgjöf/Maris Optimum/Golden\_redfish\_model/'  
 path\_grs\_weight =\  
 'R:/Ráðgjöf/Maris Optimum/Data\_from\_Hafro\_20220906/ssb\_weights/'  
  
 red\_length\_weights = pd.read\_csv(  
 path\_grs\_weight + "red\_length\_weights.csv")  
 red\_length\_weights.set\_index('ar', inplace=True)  
  
 wl\_df = pd.read\_csv(path\_grs\_str+'RED\_gadget\_n\_at\_age.csv', sep=',')  
  
 Xl = wl\_df[['year', 'mean\_length']].copy()  
 yl = wl\_df[['year', 'mean\_weight']].copy()  
  
 for index, row in Xl.iterrows():  
 Xl.at[index, 'squared'] = row[1]\*\*2  
  
 for year in range(1985, 2022):  
 average\_weight = 0  
 reg\_X = Xl[Xl['year'] == year]  
 reg\_y = yl[yl['year'] == year]  
 reg = LinearRegression().fit(  
 reg\_X[['mean\_length', 'squared']], reg\_y[['mean\_weight']])  
 b = reg.coef\_[0][0]  
 a = reg.coef\_[0][1]  
 c = reg.intercept\_  
 for col in range(1020, 1060):  
 if year > 1995 and \  
 pd.isna(red\_length\_weights.loc[year, str(col-1000)]) is False:  
 average\_weight += red\_length\_weights.loc[year, str(col-1000)]  
 else:  
 average\_weight += (X\_catch\_per\_df.loc[year, str(col)]) \* (  
 a\*(col - 1000)\*\*2 + b\*(col - 1000) + c)  
 catch\_df.at[  
 year - 1985, 'number'] = catch\_df.loc[  
 year - 1985, 'catch']/average\_weight  
  
 if show\_heatmaps:  
 df = red\_length\_weights  
 scaled\_df = (df - df.min(axis=0))/(df.max(axis=0) - df.min(axis=0))  
 fig, ax = plt.subplots(figsize=(8,6))  
 ax = sns.heatmap(scaled\_df, linewidths=.5, cmap='RdYlGn')  
 ax.set(ylabel='year')  
 plt.show()  
 return catch\_df

The data used was received from the Icelandic Marine and Freshwater Institute spring surveys and catch reports by the Directorate of Fisheries (DF). We have modified the survey data as large individual hauls may have undue influence on the overall samples and give a biased view of the size distribution in the survey. The relatitve importance of the largest catches has been scaled down to the size of the catch sizes at the 96th fractile to reduce this bias. See appendix 1 for more detail.

The regression is set up in the normal fashion: y = f(X), where y, the independant variable is the stock index and X, the dependent variable (“features”) is the set of observation and f is a function derived from the regression.

X contains 108 variables. The first 51 are taken directly from every spring survey from 1985 and are the calculated percentage of individuals that appear in each length group. The following variable is, max(cum), is the number of individials observed in the survey.

The next five variables are slack variables for years 2015 - 2019.

The final 51 variables contain the number of individuals observed in each length group by the fishing fleet as collected by the DF.

# Plots and Figures

### Heat maps

def heat\_maps(X, y):  
 """  
 Plot heatmaps.  
  
 Parameters  
 ----------  
 X : Independent variables.  
 y : Dependent variables.  
  
 Returns  
 -------  
 Pictues of heat maps.  
  
 """  
 fig, ax = plt.subplots(figsize=(8,6))  
 sns.color\_palette("vlag", as\_cmap=True)  
 ax = sns.heatmap(X.iloc[:, :50], cmap='RdYlGn')  
 ax.set(ylabel='year')  
   
 plt.show()  
   
 fig, ax = plt.subplots(figsize=(8,6))  
 ax = sns.heatmap((  
 y.to\_numpy() \* X.iloc[:, 0:50].T).T, cmap='RdYlGn')  
 ax.set(ylabel='year')  
 plt.show()

### Plot over possible stock range

The algorithm iterates over the range 343 to 453 million individuals in steps of one million individuals at a time. It finds the lowest mean absolute error at each step. The plot shows the mean absolute error and .

def plot\_result\_range(result\_dict, interval\_int, fractile, y\_offset):  
 """  
 Plot mae and r^2 for the range of the looped regression.  
  
 Parameters  
 ----------  
 result\_dict : a dictianary containing the results from different regression  
 over the interval.  
 interval\_int : integer with the increments used over the interval.  
 fractile : fractile used for Winsorization.  
  
  
 Returns  
 -------  
 None.  
  
 """  
 result\_dict['x'] = list(range(343+y\_offset,  
 453+y\_offset,  
 int(interval\_int/1e6)))  
 fig, ax = plt.subplots()  
 sns.set(style='whitegrid',  
 palette='pastel', )  
 sns.lineplot(x='x',  
 y='r2',  
 data=result\_dict,  
 color="red",  
 ax=ax)  
 ax.set(xlabel='size of stock in millions',  
 ylabel='r2, red',  
 title='school fractile:'+fractile+'\n'  
 + '1985-2022',  
 ylim=(0, 1))  
  
 ax2 = ax.twinx()  
 sns.lineplot(x='x',  
 y='mae',  
 data=result\_dict,  
 color='blue',  
 markers=True, ax=ax2)  
 ax2.set(ylabel='mean average error, blue')  
 plt.show()

### Shap calculations and plots

Shap [values](https://github.com/slundberg/shap) are calculated for the values with the lowest mean absolute error. The twenty most influential feaures are displayed. As there is a strong degree of dependency between the features (see Appendix 2 below) care must be applied to the analysis (see for example this article: [The limitations of shap](https://towardsdatascience.com/the-limitations-of-shap-703f34061d86#:~:text=But%20don't%20let%20the%20popularity%20persuade%20you.&text=We%20will%20discuss%203%20significant,them%20%E2%80%94%20we%20make%20up%20stories)).

def shap\_calculations\_xgb(regressor, XX\_df):  
 """  
 Calculate shap vales for the xgb\_regressor and the dependant variables.  
  
 Parameters  
 ----------  
 regressor : TYPE  
 DESCRIPTION.  
 XX\_df : TYPE  
 DESCRIPTION.  
  
 Returns  
 -------  
 Importance plot and waterfall plot  
  
 """  
   
 fig = plt.figure()  
 fig.set\_figheight(5)  
 explainer = shap.TreeExplainer(regressor)  
  
 shap\_values = explainer.shap\_values(XX\_df)  
  
 shap.summary\_plot(shap\_values,  
 XX\_df,  
 plot\_type="violin",  
 max\_display=15,  
 show=False)  
   
 plt.gcf().set\_size\_inches(8,6)  
 plt.show()  
   
  
 shap\_values = explainer(XX\_df)  
 shap.plots.waterfall(shap\_values[37],   
 max\_display=15,  
 show=False)  
   
 plt.gcf().set\_size\_inches(8,6)  
 plt.show()

### Regression over possible ending values of stock

The algorithm itearates over the range 343 to 453 million individuls in steps of one 1 million at a time. It finds parameters for the gradient boosting algorithm by using grid search where the root mean square error is as low as possible at each step. It choses the number of individuals based on the lowest mean absolute error value.

def regression\_over\_possible\_values\_XGB(X, y, interval\_int):  
 """  
 Loop over possible stock sizes, regressing in every step.  
  
 Parameters  
 ----------  
 X : Dataframe with dependent variables.  
 y : Dataframe with independent variables.  
 interval\_int : step interval.  
  
  
 Returns  
 -------  
 result\_dict : json string with solutions in each interval.  
  
 """  
 parameters = {  
 'eval\_metric': ["mae"],  
 'learning\_rate': [.01, .05, .1, .2, .3],  
 'max\_depth': [2],  
 'min\_child\_weight': [2],  
 'subsample': [.5],  
 'colsample\_bytree': [.5],  
 'n\_estimators': [120]  
 }  
 test\_size = .25  
 seed = 5  
 result\_dict = {'fjoldi2022': [], 'fjoldi2021': [],  
 'fjoldi2020': [], 'mae': [], 'rmse': [], 'r2': [],  
 'evs': []}  
  
 xgb1 = xgb.XGBRegressor(objective='reg:squarederror', seed=seed)  
  
 for add\_int in range(0, 110000000, interval\_int):  
 # print(add\_int)  
  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 X,  
 y,  
 test\_size=test\_size,  
 random\_state=seed)  
  
 n\_iter = 200  
 n\_iter = n\_iter  
  
 xgb\_regressor = GridSearchCV(xgb1,  
 parameters,  
 n\_jobs=-1,  
 cv=3,  
 verbose=False)  
  
 eval\_set = [(X\_train, y\_train), (X\_test, y\_test)]  
  
 xgb\_regressor.fit(X\_train,  
 y\_train,  
 eval\_set=eval\_set,  
 verbose=False)  
  
 y\_pred\_test = xgb\_regressor.predict(X\_test)  
  
 # y\_pred\_test = svr\_regression(X\_train, y\_train, X\_test, y\_test)  
  
 result\_dict['fjoldi2022'].append(y.iloc[37])  
 result\_dict['fjoldi2021'].append(y.iloc[36])  
 result\_dict['fjoldi2020'].append(y.iloc[35])  
  
 result\_dict['mae'].append(mean\_absolute\_error(y\_test,  
 y\_pred\_test))  
 result\_dict['rmse'].append(math.sqrt(mean\_squared\_error(y\_test,  
 y\_pred\_test)))  
 result\_dict['r2'].append(r2\_score(y\_test,  
 y\_pred\_test))  
 result\_dict['evs'].append(explained\_variance\_score(y\_test,  
 y\_pred\_test))  
 y.iat[35] += interval\_int \* (y.iloc[35]/y.iloc[36])  
 y.iat[36] += interval\_int \* (y.iloc[36]/y.iloc[37])  
 y.iat[37] += interval\_int  
  
 min\_value = min(result\_dict['mae'])  
 min\_index = result\_dict['mae'].index(min\_value)  
  
 y.iat[35] = result\_dict['fjoldi2020'][min\_index]  
 y.iat[36] = result\_dict['fjoldi2021'][min\_index]  
 y.iat[37] = result\_dict['fjoldi2022'][min\_index]  
  
 regressor = GridSearchCV(xgb1,  
 parameters,  
 n\_jobs=-1,  
 cv=3,  
 verbose=0)  
  
 regressor.fit(X, y)  
 params = regressor.best\_params\_  
 regressor = xgb.XGBRegressor(\*\*params)  
 regressor.fit(X, y)  
  
  
 return result\_dict, regressor

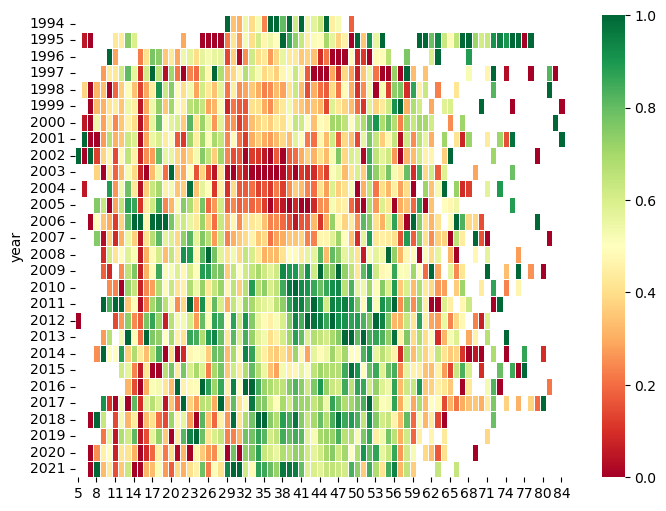
### Free regression (with slack variables for the period 2015 to 2019)

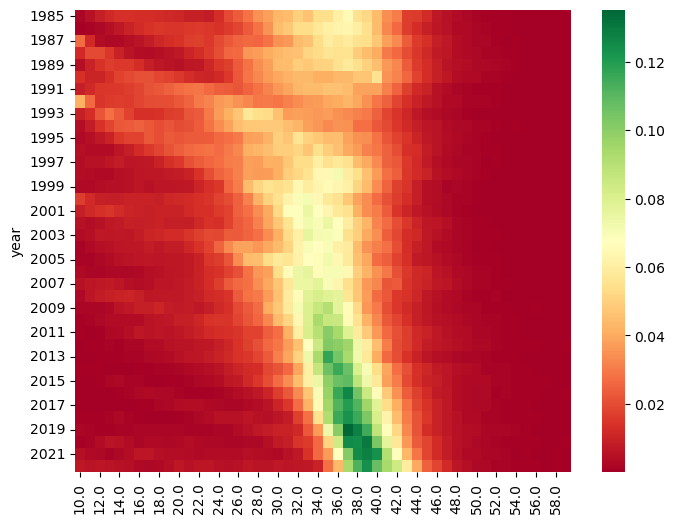
def free\_regression\_XGB(X, y):  
 """  
 Regression where part of y values are free.  
  
 Parameters  
 ----------  
 X : Dataframe with dependent variables.  
 y : Dataframe with independent variables.  
  
 Returns  
 -------  
 returns array with predection for y based intiial values of X and  
 the regression.  
  
 """  
 parameters = {  
 'eval\_metric': ["mae"],  
 'learning\_rate': [0.01, 0.1],  
 'max\_depth': [2],  
 'min\_child\_weight': [2],  
 'subsample': [.5],  
 'colsample\_bytree': [.5],  
 'n\_estimators': [120]  
 }  
  
 seed = 5  
  
 xgb1 = xgb.XGBRegressor(objective='reg:squarederror', seed=seed)  
  
 year = 29  
  
 X\_train = pd.concat([X.iloc[:year, :], X.iloc[34:, :]])  
 y\_train = pd.concat([y.iloc[:year], y.iloc[34:]])  
 # print(y\_train)  
 X\_test = X.iloc[year:34, :]  
 y\_test = y.iloc[year:34]  
  
 n\_iter = 200  
 n\_iter = n\_iter  
  
 eval\_set = [(X\_train, y\_train), (X\_test, y\_test)]  
  
 regressor = GridSearchCV(xgb1,  
 parameters,  
 n\_jobs=-1,  
 cv=3,  
 verbose=0)  
  
 regressor.fit(X\_train,  
 y\_train,  
 eval\_set=eval\_set,  
 verbose=0)  
  
 # params = regressor.best\_params\_  
 # regressor = xgb.XGBRegressor(\*\*params)  
 # regressor.fit(X\_train, y\_train)  
  
 return regressor.predict(X)

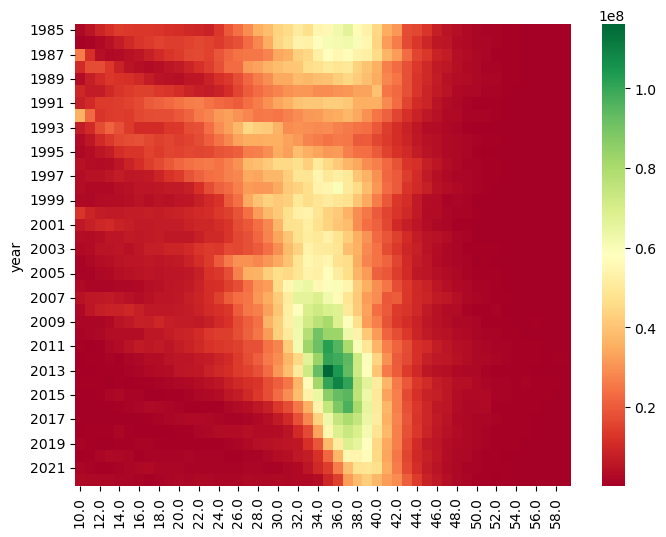
## Model run (#1)

1. Adjusted data fetched
2. Heat maps generated
3. Regression in a loop over possible values

fractile = '096'  
interval\_int = 1000000  
  
(X, y) = get\_new\_data(fractile, show\_heatmaps=True)



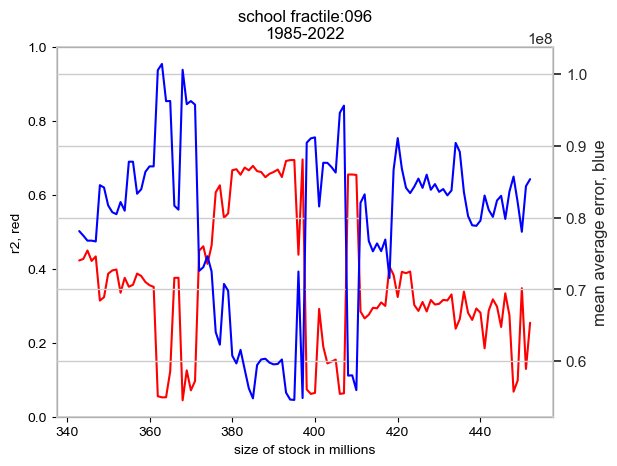




Above there are three heatmaps. in the first heat map we show the relative weight of individuals of certain length over the years. We compared the mean weight for individuals of lengths from 30-50cm for the years in this decade to the mean of the average weights of the whole population sample. Interestingly they differ significantly at the α=.01 level.

The other two heat maps show the distribution of the length classes in percentages and millions. They show really well the precarious position of the golden redfish stock.

result\_dict\_gb, regressor = regression\_over\_possible\_values\_XGB(X, y, interval\_int)  
plot\_result\_range(result\_dict\_gb, interval\_int, fractile, 0)



The optimal solutions (the solutions that best balance the mean bsolute error (“mae”) and the explainability () ) is that there are currently 385, 395 or 412 m individuals in the stock. At this point mae 52m individuals and 0.55. While this is a solid estimate we are looking for a better result in terms of r2 og mae.

## Model run (#2)

1. Adjusted data fetched
2. “Free” gradient boosting used to estimate actual stock sizes for 2015-2019
3. Regression in a loop over possible values
4. Shap analysis

(X\_observed, y\_observed) = get\_new\_data(fractile, False)  
y\_2022 = y\_observed.iloc[37]  
  
y\_calculated = pd.Series(free\_regression\_XGB(X\_observed, y\_observed))  
y\_offset = int((y\_calculated.iloc[37]-y\_2022)/1e6)  
result\_dict\_gb, regressor = regression\_over\_possible\_values\_XGB(X\_observed,  
 y\_calculated,  
 interval\_int)

plot\_result\_range(result\_dict\_gb,  
 interval\_int,  
 fractile,  
 y\_offset)

Given the rapid decline in stocks in the past few years we were keen to test if the estimates of stock sizes in the past decade may have been excessive. By introducing slack variables for the years 2015 - 2019 and fixing the test on the same period as well as fixing the result of the last 3years we got a much better result. In this case the current stock size was estimated at 378 m individuals. At this point mae 47m individuals and 0.8.

This result is much better than the previous instance. The result of IMFI of 343 m individuals falls within the confidence interval so the results are essentially the same.

In the table below we show the difference between the observed stock size and the size as calculated by the model:

y\_observed.index=range(0,38)  
y\_df = pd.concat([y\_observed, y\_calculated], axis=1, join='inner')  
y\_df.columns = ['observed', 'calculated']  
y\_df['difference'] = y\_df['observed'] - y\_df['calculated']  
y\_df = y\_df.rename(index = lambda x: x + 1985)  
pd.options.display.float\_format = '{:,.0f}'.format  
  
print(y\_df.iloc[30:38, :])

shap\_calculations\_xgb(regressor, X)

We performed two types of shap analysis an dthey are shown on the figures above (summary plot and waterfall plot)

The result of the first is shown on the summary plot (the first figure) where the dependent variables are ranked in order of importance over the whole period (1985 - 2022). The first figure shows the impact of each feature in millions of individuals. If the red colour is on the positive x-axis it indicates that more individuals of this length mean more overall individuals and vice versa. Of note is that in feature from the survey e.g. 56 blue is on the negative x axis and red on the positive side. Conversely most of the fleet catch features, 10XX show the opposite impact as expected.

The most important feature is the 19 cm feature from the surveys. This feature has always been of key importance even when our models did not manage to explain variance (<<.5) and there were higher rmse and mae errors (>100.000.000). Statistically it is easy to explain this: good recruitment means that there are a lot of individuals in the total stock (in addition to the 19cm variety). Of note is that is that 1019 has red on both sides of the of the x-axis which underlines the need to protect recruits. It is also negatively skewed.

The third feature, max(cum), is interesting in that it shows that the number of individuals caught in a survey is a very good indicator of stock size.

The rest of the important features fall into two categories: 1) Sizes 29-40 in caught the survey. 2) Sizes 35 + in caught by the fleet. The states the obvious: As stock is depleted the individuals caught in surveys and by the fleet are longer.

The waterfall plot n the second figure displays how the 2022 value estimation is arrived at by the regression model. the model starts at the average number of 786 m individuals (the average of stock suveys for the years 1985 - 2022) which is shown at the bottom of the figure. It then calculates the impact of the observed number of individuals in the spring survey in 2022 as well as the estimated fleet catches. The final result, 378 m, is shown at the top of the figure and how the model got there.

### Appendix 1: Adjusting for large catches in surveys

Golden redfish lives in large schools.

|  |
| --- |
| The figure shows the age distribution without correcting for large catches |

|  |
| --- |
| The figure shows the age distribution where 96% fractile correction has been employed. Fractile levels were calibrated by examining different fractile levels between 90 and 100%, |

### Appendix 2 PCA analysis and depenency (linear or non-linear)

scaler = StandardScaler()  
X\_sca = scaler.fit\_transform(X)  
pca = PCA(n\_components=18)  
pca.fit(X\_sca)  
  
pd.options.display.float\_format = '{:,.2f}'.format  
evr = pd.DataFrame(pca.explained\_variance\_ratio\_, columns=['variability'])  
print(evr.iloc[0:5, 0:1])

The above tables illustrates that by decomposing the dependence matrix and ordering the eigenvectors by eigenvalues that almost 40% of the variability is connected to one eigenvector and that 70% of the variability can be reduced to three eigenvectors. Given that the number of dependant variables is 108 it is clear that there is a high degree of dependency between the dependant variables. Gradient boosting strength is that it discards features with the same information in already chosen features. The shap analysis above shows that the 10-15 most important features provide enough information for the model to function.