Using FishSET in the RConsole

Contents

1 Welcome to FishSET

This chapter details the goals of FishSET, outlines the different chapters of this document, and provides instructions for installing FishSET.

1.1 Introduction

The Spatial Economics Toolbox for Fisheries (FishSET) is a set of statistical programming and data management tools developed to achieve the following goals:

1. Standardize data management and organization.
2. Provide easily accessible tools to enable location choice models to provide input to the management of key fisheries.
3. Organize statistical code so that predictions of fisher behavior developed by the field's leading innovators can be incorporated and transparent to all users.

The FishSET package provides functions for:

- Data management
- Data analysis
- Visualizing data
- Mapping fishing effort
- Statistical modeling of fisher behavior
- Policy comparisons
- Reproducibility

FishSET is designed for reproducibility.

- All input and output are stored in the FishSETFolder directory and organized by projects.
- Data is stored in *fishset_db* SQLite databases, the FishSET database.
- Function calls and notes are stored in a *src* folder.
- Output (plots, table, messages, notes) are saved in an *Output* folder.

This document contains a brief description of the FishSET functions and assumes familiarity with R. If you are unfamiliar with R, we recommended running FishSET functions in the FishSET GUI rather than the RConsole. See the FishSET GUI vignette for more information.

A more detailed manual is available at the ***FishSET GitHub site***.

1.2 Getting started

If not all ready installed, install R and RStudio. RStudio is not required but is recommended.

To install the FishSET package, all users need the `devtools` package and Windows users must install Rtools.

Install devtools (required) `install.packages("devtools")`

```
library(devtools)
```

FishSET is provided as a compressed file that can be installed directly from the FishSET GitHUB site (recommended) or from a locally saved file location.

Install FishSET To install FishSET from GitHUB

```
devtools::install_github("name/FishSET")
```

To install FishSET from local file directory `devtools::install_local("PATH/TO/Directory/Containing/FishSET")`

```
library(FishSET)
```

Dependencies should be automatically installed. This can take awhile.

For more detailed directions and images, see the Installing R and RStudio and Installing FishSET section of the FishSET Manual.

##Using FishSET FishSET functions can be called in the R console or through the FishSET Graphical User Interface (FishSET GUI). This document details how to use FishSET in the R Console. The FishSET GUI is detailed in the *FishSET GUI* vignette. Background and greater detail on functions can be found in the FishSET Help Manual.

1.3 What is a project?

Data and output are organized into *projects*. A `project` is a unique identifier for a set of data and analyses. `Project` names can be any set of alpha-numeric characters. For instance, a `project` can be called *pollock* or *pollock2020*. All input and output are saved to a directory with the name of the `project` within the FishSETFolder directory.

1.4 Chapter overview

1.4.1 Chapter 2 Data

This chapter goes over the types of data used in FishSET and how to load the data into FishSET. The chapter then outlines functions for conducting basic data quality checks and fixing any issues that arise.

1.4.2 Chapter 3 Data Exploration

This chapter goes over the functions to visualize and explore the data.

1.4.3 Chapter 4 Spatial Functions

This chapter outlines the numerous functions related to visualizing the data spatially and generating plots of the spatial distribution of the data.

1.4.4 Chapter 5 Creating and Modifying Data

This chapter outlines the functions to generate new variables, such as catch per unit effort, or modify existing variables, such as extracting year from a date variable.

1.4.5 Chapter 6 Modeling Functions

This chapter goes over the steps required to generate the data used in the models, define the models, run the models, and compare output.

1.4.6 Chapter 7 Policy and Welfare Analysis

The chapter goes over how to define policy scenarios and run welfare analyses.

1.4.7 Chapter 8 Reporting

This chapter goes over how to open and use the report template.

2 Data

2.1 Required and optional data

FishSET uses five data types.

File Type	Description
Primary data	Required. Contains the main data for the model.
Port data	Required. Contains latitude and longitude locations of ports.
Spatial data	Required. File defining boundaries of fishery or regulatory zones.
Gridded data	Optional. Contains additional data that varies by two dimensions. Example, sea surface temperature measure
Auxiliary data	Optional. Contains additional data to link to the primary data. Example, vessel characteristics.

All optional data files must be linkable to the primary data file.

Gridded data should link to the fishery or regulatory areas in the spatial data. Auxilary data must link to a variable in the primary data table.

FishSET will accept several data formats including:

- csv files
- geojson files
- json files
- matlab files
- R files

- shape files
- spss files
- stata files
- txt files
- xls/xlsx files

Details on the different data types are detailed in the Data Types section of the FishSET manual.

2.2 Loading and reading data

2.2.1 Loading data to the FishSET database

All data should be saved to the FishSET database. Doing so ensures that data for a project are stored together.

The *load* functions check that the data file can be saved to the FishSET database and, if all checks pass, saves the data file to the FishSET database.

When the primary data is saved, two versions of the data are created, a raw table with the original data and a working table on which all analyses will be conducted. The working table will also be available in the global environment. Revised data can be saved as interim tables in the database along with a “finalized” data table (data table used for models).

2.2.1.1 Functions

`load_maindata()`	Primary data file.
`load_port()`	Port data file.
`load_aux ()`	Auxiliary data file.
`load_grid()`	Gridded data file.

2.2.1.2 Function details `load_maindata(dat, project, over_write = TRUE, compare = FALSE, y = NULL)` read in a data file, then parses it, and finallys saves the data table to the FishSET database if no errors are found. The function calls the `data_verification` function to check for common data quality issues and that latitude and longitude are defined in `dat`. If no issues are found, the data is saved in the FishSET database for the `project` as `projectMainDataTable` and loaded into the working environment. If `over_write =TRUE`, the existing data table for the project, if exists in the FishSET database, will be overwritten. If `compare = TRUE` the `fishset_compare` function is called and `dat` is compared with `y`, a previously saved version of `dat`.

`load_port`

`load_spatial`

`load_aux`

`load_grid`

> "Add details about loading port, spatial , auxiliary, and gridded files."

2.2.2 Reading data into R

Data can also be read into R without saving it to the FishSET database. This function should only be used to view and evaluate data tables. If the data is going to be used in FishSET it should be loaded into the FishSET database.

To read data files into the working environment use:

```
read_dat(x, data.type = NULL, is.map = F, ...).
```

2.2.2.1 Function details `read_dat(x, data.type=NULL, is.map = FALSE, drv = NULL, dbname = NULL, user = NULL, password = NULL, ...)` import data from local file directory or webpage into the R environment. `x`is the name and path, or web address, of the dataset to be read in. `data.type` is the file extension. It is optional as the function will attempt to detect the file extension. `data.type` must be defined if `x` is the path to a shape folder, if `x` is a Google spreadsheet (`data.type = 'google'`), or if the correct extension cannot be derived from `x`. Set `is.map = TRUE` only for spatial files in a .json file extension. For sql files, additional arguments may be required, including the database driver (`drv`), the database name (`dbname`), and user name (`user`) and password (`password`) for the SQL database.

`read_dat()` is flexible and will accommodate several data extensions and additional format-specific arguments. It is a wrapper function that calls a file extension-specific function to read in the data. `x` is the name and path of the dataset to be read in. `data.type` is optional argument to specify the file extension. If `data.type` is not specified, the detected data extension in `x` will be used. `is.map` is a logical argument specifying if `x` is a spatial data file. is.map must be set to TRUE if a spatial data file is not in a geojson or shape file. `...` is used to add additional extension-specific arguments. See function documentation for more details.

2.2.3 Loading data from FishSET database into the the working environment

Data saved to the FishSET database can be reloaded into the working environment using

```
load_data(project, name=NULL).
```

Specify the project name (`project`) and, optionally, the table name (`name`). `name` is optional and is used to specify a specific version of the data table. For example,

```
load_data("pollock", name="pollockMainDataTable20200101")
```

loads the primary data from project *pollock* that was saved on January 1st, 2020. If `name` is NULL, the undated working table is pulled.

```
##Load in the data from the pollock project
library(FishSET)
load_data("pollock")
```

2.3 Merging gridded and auxiliary data with the primary data

Secondary data (auxiliary, port, gridded) can be merged into the primary dataset. The default method to merge is to use a left join; all columns and rows from the primary dataset are kept whereas only matching columns and rows from the secondary table are joined. Although all secondary data types can be merged into the primary day, generally only auxiliary data (such as vessel characteristics) will need to be merged.

2.3.0.1 Functions

```
merge_dat()   Merge secondary data into the primary data tahble
split_dat()   Split and save secondary data from the primary data table
```

2.3.0.2 Function details `merge_dat(dat, other, project, main_key, other_key, other_type = NULL, merge_type = "left")` merges secondary data (`other`) to the primary data (`dat`) based on one or more variables for `dat` (`main_key`) and `other` (`other_key`). If more than one variable is being used for merging, include as a list. Variable names do not have to be the same between `main_key` and `other_key` but they must be included in the same order. `other_type` is optional and defines the data type of the secondary data ("aux", "grid", "spat", "port"). `merge_type` can be "left" or "full". "left" keeps all columns and rows in `dat` but only the rows from `other` in which the column values match column values in `dat`. "full" keeps all rows of both `dat` and `other`. "left" merge is preferred as using "full" can result in a larger frequency of missing data if the secondary data contains observations beyond the scope of `dat`. For example, `dat` contains catch data only on catcher vessels but other contains vessel characteristics for catcher vessels, mothership vessels, and catch processor vessels.

`split_dat(dat, project, aux = NULL, split_by = NULL, key, output = "main")` separates secondary data (auxiliary, gridded, or port data) from the MainDatatable. Identify data to remove from `dat` using an existing secondary data table or list of column names. If `aux` is specified, the column names in `dat` that match the column names in `aux` are separated from `dat`. If `aux = NULL` and `split_by` is a list, then column names in `dat` matching `split_by` are separated from `dat`. `key` is the columns that link `dat` and `aux`. If `aux` is null then `key` should be a column name from `split_by`. The function can return just the primary data (`output = "main"`), just the separated table (`output = "aux"`), or both, in a list (`output = "both"`).

2.4 Working in the FishSET database

The functions in this section interface with the FishSET database to display the requested output in the R Console. The functions are wrappers for functions in the **DBI** package. They allow users to focus on the data and not on managing the database or learning SQL.

Basic functions:

```
tables_database(project)        View list of all tables in the FishSET database for the project.
list_MainDataTables(project)      View list of MainDataTables in FishSET database for the project.
list_PortTables(project)          View list of PortTables in FishSET database for the project.
list_logs(project)                View list of all log files for the project.
table_fields(table, project)    View fields in specified table for the project.
table_view(table, project)      Display specified table for the project.
table_exists(table, project)      Check if table exists in the FishSET database for the project.
table_remove(table, project)      Remove table from the FishSET database for the project.
```

Model output files can be viewed in the FishSET database using:

```
globalcheck_view(table, project)  View error output discrete choice models for the project
model_out_view(table, project)    Load discrete choice model output for the project
```

2.5 Metadata

"This needs to be detailed"

2.6 Data management functions

Data should, at a minimum, be checked for entry errors and errors resulting from data import and conversion. Users should also investigate and summarize the data to understand potential data and statistical problems.

FishSET provides a number of statistical and graphic functions for evaluating data quality. These functions focus on issues that are most likely to affect model convergence and model performance. They do not cover all potential data quality issues that exist in your data.

2.6.1 Functions

2.6.1.1 Visualizing the data

```
summary_stats()    Table with basic summary statistics.
spatial_hist()     Histogram of lat and lon by grouping variable.
spatial_summary()  Line plots of selected variable against date and zone.
map_plot()         Static map of haul locations.
```

2.6.1.2 Subsetting the data

```
select_vars()      Select variables to add back into the working dataset.
add_vars()         Add variables from `select_vars` into primary dataset.
```

2.6.1.3 Checking for outliers

`outlier_boxplot(dat, project, x = NULL)` returns a box-and-whisker plot of all numeric variables in `dat`. Use `x` to define a set of variables to assess. This function helps to identify which variables potentially contain outlier variables and should be run before outlier_table() and outlier_plot() functions, which can be applied to only one variable at a time.

`outlier_table()`	`Returns table with quantiles for all numeric variables in the data table.`
`outlier_plot()`	`Returns a plot of the selected variable.`
`outlier_remove()`	`Returns modified data set where outliers have been removed.`

2.6.1.4 Checking for empty variables, unique rows, NAs, and NaNs

`empty_vars`	`checks for emtpy variables.`
`nan_identify()`	`Checks whether NAs or NaNs are present in any variable.`
`nan_filter()`	`Returns modified data set where NaNs have been removed or replaced.`
`na_filter()`	`Returns modified data set where NAs have been removed or replaced.`
`unique_filter()`	`Check for and remove duplicate rows.`

2.6.1.5 Filtering rows of the dataset

`filter_table()`	`Stores filter functions in a table.`
`filter_dat()`	`Apply stored and new filter functions to data.`

2.6.1.6 Converting class or lon/lat units

`changeclass()`	`Convert variable class.`
`degree()`	`Convert lon/lat to decimal format.`

2.6.1.7 Wrappers for multiple "checking" functions

`data_check()`	`Wrapper for data quality check functions, including data_verification().`
`data_verification()`	`Checks for unique column names, each row is a unique choice occurrence, etc.`
`check_model_data()`	`Checks for presence of NAs, NaNs and Inf in model data. Function must be run befor`

2.6.2 Function details

2.6.2.1 Visualizing the data

`summary_stats(dat, project, x = NULL)` prints summary statistics for each variable in the `dat`. If `x` is specified, summary stats will be returned only for that variable.

```
spatial_hist()
spatial_summary()
map_plot()
```

2.6.2.2 Subsetting the data

```
select_vars()
add_vars()
```

2.6.2.3 Checking for outliers

```
outlier_boxplot
outlier_table()
outlier_plot()
outlier_remove()
```

2.6.2.4 Checking for empty variables, unique rows, NAs, and NaNs `empty_vars_filter(dat, project, remove = FALSE)` checks for empty variables and prints an outcome message to the console. If empty variables are present and `remove = TRUE`, then empty variables will be removed from `dat`.

```
nan_identify()
nan_filter()
```

`na_filter(dat, project, x=NULL, replace = F, remove = F, rep.value = NA, over_write = FALSE)` checks for NAs and removes or replaces NAs. To check for NAs across `dat`, run the function specifying only `dat` [`na_filter(pollockMainDataTable)`]. The function will return a statement of which variables, if any, contain NAs. To remove NAs, use `remove = TRUE`. All rows containing NAs in variable(s) `x` will be removed from `dat`. To replace NAs, use `replace = TRUE`. If `replace = TRUE` and `rep.value` is not defined, then NAs are replaced with the mean value of `x`. `rep.value` must be specified if `x` is not numeric. The function returns the modified dataset if either `replace` or `remove` are true. Save the modified data table to the FishSET database by setting `over_write = TRUE`.

`unique_filter(dat, project, remove = FALSE)` returns a statement of whether each row in `dat` is unique. Set `remove=TRUE` to remove duplicate rows.

2.6.2.5 Filtering rows of the dataset

```
filter_table()
filter_dat()
```

2.6.2.6 Converting variable class or lon/lat units `changeclass(dat, project, x = NULL, newclass = NULL, savedat = FALSE)` returns a table with data class for each variable in `dat` and changes variable classes. To view variable classes run the function with default settings, specifying only `dat` and `project`. If variable class should be changed, run the function again, specifying the variables (`x`) to be change and the newclasses (`newclass`). Length of `newclass` should match the length of `x` unless all variables in `x` should be the same class. Options are “numeric”, “factor”, “date”, and “character”. Use `savedat = TRUE` to save modified data table.

```
degree()
```

2.6.2.7 Wrappers for multiple “checking” functions `data_check()` is the primary function to check for data quality issues in the dataset. The function calls `summary_stats`, `nan_identify`, `outlier_table` and `outlier_plot`, and `data_verification`, which contains script to check for unique column names and empty columns, that each row is a unique observation at haul or trip level, and calls the `degree` function. Use `outlier_remove`, `na_filter`, `nan_filter`, and `degree` to correct any errors. Empty columns and duplicate rows will be removed.

`data_verification(dat, project)` checks that all column names in the `dat` are unique, whether any columns in the data frame are empty, whether each row is a unique choice occurrence at the haul or trip level, and that either latitude and longitude or fishing area are included.

```
#data_check(pollockMainDataTable, 'pollock', 'OFFICIAL_TOTAL_CATH_MT')
```

`check_model_data` checks for data quality issues that will result in errors when running models, including NAs, NaNs and INF values in the dataset and that each row is a unique occurrence at the haul or trip level. Run this function before creating the model design file (`make_model_design`). Even if the data passed the `data_check` tests, errors may be produced by data modification and data creation functions.

Use `filter_table` and `filter_dat` to subset the data, retaining rows that meet the condition. Define and save, for records and future use, conditions in `filter_table`. Apply filters with `filter_dat`.

Data table	Vector	FilterFunction
MainDataTable	"PORT_CODE"	PORT_CODE == 1
MainDataTable	"TRIP_START"	TRIP_START >= 2011-02-01
PortDataTable	"LATITUDE"	LATITUDE < 57

3 Data Exploration

Several functions are available for summarizing and visualizing data.

We group exploratory data functions into 1) basic exploratory analysis - functions exploring the distribution, availability, and variance of data, 2) fleet summaries – functions to view and understand fleets and identify meaningful groupings, and 3) simple analyses - functions to evaluate relationships between variables and identify redundant variables.

All output is saved to the output folder.

3.1 Data Exploration functions

Basic exploratory analysis

```
map_kernel()       Spatial kernel density plot.
getis_ord_stats() Getis-Ord statistic.
morans_stats()     Moran's I statistic.
map_plot()         Static map of haul locations.
map_viewer()       Interactive map of haul locations or paths with zone.
spatial_summary() View aggregated variable by date and fishing zone.
spatial_hist()     Assess spatial variance/clumping of grouping variable.
temp_plot()        View distribution of variable over time.
```

Fleet summaries

```
vessel_count()     Counts of unique vessels active within a specified period.
species_catch()    Aggregates total catch for one or more species.
weekly_catch()     Calculates weekly catch for one or more species.
weekly_effort()    Generates the average CPUE by week.
density_plot()     Plots the chosen density plot of chosen variable.
bycatch()          Compares mean CPUE and share (or count) of total catch by time period.
roll_catch()       Rolling catch (specify window size and summary statistic).
trip_length()      View trip duration.
```

Simple analyses

`corr_out()`	`View the correlation coefficient between numeric variables.`
`xy_plot()`	`Asses fitted relationship between two variables.`

All tables and plots generated are saved to the output folder.

###Function details

`map_kernel(dat, project, type, latlon, group, facet, date, filter_date, filter_value, minmax)` returns the kernel density plot in the specified plot `type` (*"point"*, *"contours"*, *"gradient"*) based on the specified latitude and longitude variables (`latlon`). All other arguments are optional. `group` is the variable in `dat` containing group identifiers. If groups are specified, users can map each group as a separate facet if `facet = TRUE`. Data can be filtered by a `date` variable. Select how to filter using `filter_date` and values with `filter_value`. `filter_date` options are *"year"*, *"month"*, and *"year-month"*. `filter_value` should be four digits if year and 2 digits if month. Use colon for a range. Use a list if using *"year-month"*, with the format: *list(year(s), month(s))*. For example, *list(2011:2013, 5:7)*. Use `minmax` to limit the map extent. `minmax` is a vector of length four corresponding to c(minlat, maxlat, minlon, maxlon).
`getis_ord_stats(dat, project, varofint, spat, lon.dat, lat.dat, cat, lon.grid, lat.grid)` is a wrapper function to calculate global and local Getis-Ord (the degree, within each zone, that high or low values of the `varofint` cluster in space) by discrete area. Function utilizes the `localG` and `knearneigh` functions from the **spdep** package. The spatial input is a row-standardized spatial weights matrix for computed nearest neighbor matrix, which is the null setting for the `nb2listw` function. Requires a data frame with area as a factor, the lon/lat centroid for each area, the lon/lat outlining each area, and the variable of interest `varofint` or a spatial data file with lon/lat defining boundaries of area/zones and variable of interest for weighting. If the centroid is not included in the spatial data file, then `find_centroid()` can be called to calculate the centroid of each zone. If the variable of interest is not associated with an area/zone then the `assignment_column()` function can be used to assign each observation to a zone. Arguments to identify centroid and assign variable of interest to area/zone are optional and default to NULL.
`morans_stats(dat, project, varofint, spat, lon.dat, lat.dat = NULL, cat, lon.grid, lat.grid)` is a wrapper function to calculate global and local Moran's I (degree of spatial autocorrelation) of a variable (`varofint`) by discrete area. Function utilizes the `localmoran()` and `knearneigh()` functions from the **spdep** package. The spatial input is a row-standardized spatial weights matrix for computed nearest neighbor matrix, which is the null setting for the `nb2listw()` function. The function requires a spatial data file with latitude and longitude defining boundaries of area/zones. If zonal centroid is not included in the spatial data file, then FishSET's `find_centroid()` function is called. If each observation in `dat` is not associated with an area/zone, then FishSET's `assignment_column()` is called to assign each observation to a zone. Arguments to identify zonal centroids and assign variable of interest to area/zone are optional and default to NULL.
`map_plot(dat, project, lat, lon, minmax, percshown)` plots observed vessel locations on a map. `lon` and `lat` are the names of variables containing longitude and latitude data in dat. If the predefined map extent needs adjusting, set limits with `minmax = c(minlat, maxlat, minlon, maxlon)`. To show a random subset of points, set `percshown` to the percent of points to show. Please consider any confidentiality concerns before sharing or publishing the map. This graphic return a static but shareable map. See `map_viewer()` to view a dynamic map with greater functionality.
`map_viewer(dat, gridfile, avd, avm, num_vars, temp_vars, id_vars, lon_start, lat_start, lon_end, lat_end)` opens an interactive map in your default web browser to view vessel points or maps. `gridfile` is required to overlay fishery management/regulatory zones on the map. `dat` and `gridfile` are linked through `avd` (variable in `dat` containing zone name or identifier) and `avm` (property in `gridfile` containing zone name or identifier). Vessel haul points or vessel haul paths can be plotted. `lon_start` and `lat_start` are the variable names in `dat` containing the longitude and latitude of the vessel location points to plot or starting location if plotting paths. `lon_end` and `lat_end` are the ending latitude and longitudes if plotting a path. Leave as NULL if plotting location points. Other plotting and grouping arguments include `num_vars` (a list of one or more numeric variables in `dat`), `temp_vars` (a list of one or more temporal variables in `dat`), and `id_vars` (a list of categorical variables in `dat`). These variables are used to create additional scatter plots for assessing the data. Multiple numeric, temporal, and ID variables can be included but only one will be shown at a time. To close the server connection run `servr::daemon_stop()` in the

console. Note that it can take up to a minute for the data to be loaded onto the map. The map cannot be saved.
`spatial_summary(dat, project, stat.var, variable, gridfile, lon.grid, lat.grid, lon.dat, lat.dat, cat)` returns the variable aggregated by `stat.var` plotted against date and zone. `gridfile`, `lon.grid`, `lat.grid`, `lon.dat`, `lat.dat`, and `cat` are not required if zone assignment variable exists in `dat`.

`stat.var` options

```
length         Number of observations
no_unique_obs  Number of unique observations
perc_total     Percent of total observations
mean           Mean
median         Median
min            Minimum
max            Maximum
sum            Sum
```

`spatial_hist(dat, project, group)` returns a histogram of observed lon/lat split by grouping variable (`group`). The function is used to assess spatial variance/clumping of selected grouping variable.

`temp_plot(dat, project, var.select, len.fun, agg.fun, date.var)` returns two plots of selected variable `var.select` over time (`date.var`). The first plot can show the number of observations (`len.fun = "length"`), number of unique observations (`len.fun = "unique"`), percent of options (`len.fun = "percent"`) of `var.select` by time. The second plot shows `var.select` aggregated by `agg.fun` against date. `agg.fun` options are *mean*, *median*, *min*, *max*, and *sum*.

Fleet and group summaries

Each function has a different objective but they all have a set of optional arguments that allow for grouping the data (`group`) or filtering by date (`filter_date`) or another value (`filter_value`). These options default to NULL and do not have to be specified. There are additional options to adjust the appearance of the output. These options are detailed first; then the functions are detailed.

Additional options `group` is the name of the fleet ID variable or other grouping variable(s). Multiple group variables can be included but only the first two will be shown. For most variables, grouping variables can be merged into one variable using `combine`; in this case any number of variables can be joined, but no more than three is recommended. For functions that do not have the `combine` argument, group variables will be automatically combined if three or more are included.

`date` is the variable name from `dat` used to subset and/or facet the plot by.

`filter_date` is the type of filter to apply to the table. Options include *"date_range"*, *"year-day"*, *"year-week"*, *"year-month"*, *"year"*, *"month"*, *"week"*, or *"day"*. The *"date_range"* option will subset the data by two date values entered in `date_value`. The argument `date_value` must be provided.

`date_value` is the value to filter `date` by. If using `filter_date = "date_range"`, `date_value` should contain a start and end date, e.g. c("2011-01-01", "2011-03-15"). Otherwise, use integers (4 digits if *year*, 1-2 digits if *day*, *month*, or *week*). Use a list if using a two-part filter, e.g. "year-week", with the format `list(year, week)` or a vector if using a single period, `c(week)`. For example, `list(2011:2013, 5:7)` will filter the data table from weeks 5 through 7 for years 2011-2013 if `filter_date = "year-week"`. `c(2:5)` will filter the data February through May when `filter_date = "month"`.

`filter_by` is the variable name to filter `dat` by.

`filter_value` is a vector of values in the `filter_by` variable to filter `dat`.

`filter_expr` a valid R expression to filter `dat` by using the `filter_by` variable.

`facet_by` is the variable name to facet by. This can be a variable that exists in the dataset, or a variable created by the function such as *"year"*, *"month"*, or *"week"*. For faceting, any variable (including ones listed in `group`) can be used, but *"year"* and *"month"* are also available. Currently, combined variables cannot be faceted.

`tran` is the name of function to transform variable by. Options include *"identity"*, *"log"*, *"log2"*, *"log10"*, and *"sqrt"*. *"identify"* (no transformation) is the default.

`scale` is passed to facet_grid defining whether x- and y-axes should be consistent across plots. Defaults to *"fixed"*. Other options include *"free_y"*, *"free_x"*, and *"free"*.

`position` The position of the grouped variable for the plot. Options include *"identity"*, *"stack"*, and *"fill"*.

`output` display as *"plot"*, *"table"*, or both (*"tab_plot"*). Defaults to both.

`format_tab` defines how table output should be formatted. Options include *"wide"* (the default) and *"long"*.

Function details

`vessel_count(dat, project, v_id, date, period, group, filter_date, date_value, filter_by, filter_value, filter_expr, facet_by, combine, position, tran, value, type, scale, output)` aggregates the number (or percent) of active vessels by time period (`period`) using a column of unique vessel IDs (`v_id`). `period` options include *"year"*, *"month"*, *"weeks"* (weeks in the year), *"weekday"*, *"weekday_abv"*, *"day"* (day of the month), and *"day_of_year"*.
`species_catch(dat, project, species, date, period, fun, group, filter_date, date_value, filter_by, filter_value, filter_expr, facet_by, type, conv, tran, value, position, combine, scale, output, format_tab)` aggregates catch by time period (`period`) and species using one or more columns of catch data (`species`). `fun` is the name of the function to aggregate catch by. Defaults to *"sum"*. Use `conv` to convert catch that is in pounds to *"tons"* or *"metric_tons"*. Set `conv = "none"` if no conversion is desired. Alternative, `conv` can be a user-defined function. This is useful if catch is not in pounds.
`roll_catch(dat, project, catch, date, group, k = 10, fun, filter_date, date_value, filter_by, filter_value, filter_expr, facet_by, scale, align, conv, tran, output, ...)` returns a rolling window time series aggregation of catch that can be combined with grouping variable(s) (`group`). `catch` can be one or more columns of catch data. `k` defines the window size and `fun` is the name of the function to aggregate catch by. Defaults to *"mean"*. `align` indicates whether results of window should be left-aligned *("left")*, right-aligned *("right")*, or centered *("center")*. Defaults to *"center"*. Use `conv` to convert catch that is in pounds to "tons" or "metric_tons". Alternative, conv can be a user-defined function. This is useful if catch is not in pounds.
`weekly_catch(dat, project, species, date, fun, group, filter_date, date_value, filter_by, filter_value, filter_expr, facet_by, type, conv, tran, value, position, combine, scale, output, format_tab)` aggregates catch by week using one or more columns of catch data (`species`). `fun` is the name of the function to aggregate catch by. Defaults to *"sum"*. Use `conv` to convert catch that is in pounds to *"tons"* or *"metric_tons"*. Alternatively, `conv` can be a user-defined function. This is useful if catch is not in pounds.
`weekly_effort(dat, project, cpue, date, group, filter_date, date_value, filter_by, filter_value, filter_expr, facet_by, tran, combine, scale, output, format_tab)` calculates mean CPUE by week. This function doesn't calculate CPUE; the CPUE variable must be created in advance. One or more CPUE variables can be included.
`bycatch(dat, project, cpue, catch, date, period, names, group, filter_date, date_value, filter_by, filter_value, filter_expr, facet_by, tran, value, combine, scale, output, format_tab)` compares the average CPUE and catch total/share of total catch between one or more species. If including multiple species, `cpue`, `catch`, and `names` should be strings and species should be listed in the same order. `names` is an optional string of species names that will be used in the plot. If `names = NULL`, then species names from catch will be used. `period` is the time period to aggregate by. Options include *"year"*, *"month"*, and *"weeks"*. `value` defines whether to return raw catch *("raw")* or share of total catch *("stc")*.
`trip_dur_out(dat, project, start, end, units, catch, hauls, group, filter_date, date_value, filter_by, filter_value, filter_expr, facet_by, type, bins = 30, density, scale, tran,`

`pages, remove_neg, output, tripID, fun.time, fun.numeric)` calculates vessel trip duration in the desired unit of time given start and end dates. Obsevations with a negative trip duration can be removed from output using `remove_neg = TRUE`. If data is not at a trip level (each row is a unique trip), specify trip identifiers (`tripID`) and how to collapse data (`fun.time` and `fun.numeric`). You can also calculate CPUE and hauls per unit of time by specifying `catch` and `hauls`. Plot output can be a histogram (`type = "hist"`) or a frequency polygon (`type = "freq_poly"`). Plot output can be combined on a single page (`pages = "single"`) or printed on individual pages (`pages = "multi"`).
`density_plot(dat, project, var, type, group, date, filter_date, date_value, filter_by, filter_value, filter_expr, facet_by, tran, scale="fixed", bw, position)` returns a kernel density estimate, cumulative distribution function, or empirical cdf of selected variable. `type` options are *"kde"*, *"cdf"*, and *"ecdf"*. The `group` and `tran` arguments are not applied to the CDF plot.

View relationships

`corr_out(dat, project, variables, ...)` returns the correlation coefficient between numeric variables in `dat`. `variables` can be all numeric variables in `dat` (`variables = "all"`) or a list of column names of selected variables. Additional arguments can be added, such as the method to calculate correlation coefficients. The function defaults to displaying the Pearson correlation coefficient. To change the method, specify `method = 'kendall'` or `method = 'spearman'`.
`xy_plot(dat, project, var1, var2, regress = FALSE)` returns the fitted relationship between two variables. `var1` and `var2` are names of variables in `dat`. `regress = TRUE` returns the plot with the fitted linear regression line.

4 Spatial Functions

4.1 Mapping functions

There are several functions to view the data spatially.

`map_kernel()`	Kernel density (hotspot).
`map_plot()`	Static map of haul locations without zones.
`map_viewer()`	Interactive map of haul locations or paths with zones.

5 Creating and Modifying Data

5.0.1 Data transformation and derivation functions

Several functions exist that allow users to transform variables or derive new variables and generate specific matrices. These functions are useful to ensure date variables are in the necessary format, to generate rate variables (such as catch/hours fished), calculate more complex variables such as trip distance, and adding variance to confidential data.

Data transformation

`set_quants()`	Coded variable based on the quantiles.
`group_perc()`	Within-group percentage.
`group_diff()`	Within-group lagged difference.
`group_cumsum()`	Within-group running sum.

Dummy

`dummy_num()`	Vector of TRUE or FALSE, based on condition, the length (rows) of the data.
`dummy_matrix()`	Matrix with same dimensions at the data set filled with TRUE or FALSE.

Nominal ID

`ID_var()`	Generates a nominal variable to indicate distinct hauls or trips.
`create_seasonal_ID()`	Generates a fishery season identifier (TRUE/FALSE).
`fleet_table()`	Creates and saves a table of fleet conditions for fleet assignment.
`fleet_assign()`	Creates a fleet assignment variable based on saved fleet table.

Arithmetic

`create_var_num()`	Defined arithmetic function of two variables.
`cpue()`	Catch per unit effort.

Spatial

`assignment_column()`	Assign hauls to fishery or regulatory zones.
`create_dist_between()`	Distance between lon/lat points, port, and/or centroid of fishing zone/area.
`create_mid_haul()`	Returns the midpoint location (lon/lat) for each haul.
`create_startingloc()`	Returns vector of zone/area at point when choice of where to go next was made. Used with logit_correction likelihood.

Temporal

`temp_mod()`	Transform date variable into desired units (year, month, month/day, minutes).
`create_duration()`	Duration of time between two temporal variables based on defined time format.

Trip level

`haul_to_trip()`	Collapse the data table from haul to trip.
`trip_distance()`	Summed trip distance defined by start and end ports and hauls in between.
`trip_length()`	Computes trip duration or hauls per trip.

All functions require that `dat`, the name of the primary dataset, be specified. Use quotes if the dataset is pulled from FishSET database. Most functions in this section will add a new variable to the primary dataset. The column name of the new variables is defined with `name`. The column name defaults to the function name if `name` is not specified. Spatial functions require a spatial dataset.

###Function details

5.0.1.1 Arithmetic `create_var_num(dat, x, y, method, name)` creates a new numeric variable based on the defined arithmetic expression. `x` and `y` are names of numeric variables. `method` is an arithmetic expression such as addition, subtraction, multiplication, and division.
`cpue(dat, xWeight, xTime, name)` creates the catch per unit effort variable. `xWeight` is the name of the variable containing the catch data as a weight (pounds, tons, metric tons). `xTime` is the duration of time and must be in weeks, days, hours, or minutes. Use `create_duration()` to transform a date variable into a duration of time variable. If group-specific cpue is required, cpue must be calculated for each species.

5.0.1.2 Data transformation

`group_perc(dat, project, id_group, group, value, name, create_group_ID, drop_total_col)` creates a within-group percentage variable using primary (`id_group`) and secondary (`group`) groups. `value` is the name of a numeric variable to be used to calculate percentages. The `group` percentage is calculated as the sum of `value` across `group` (group_total) divided by the sum of `value` across `id_group` (total_group). `drop_total_col` is logical and defines whether or not to add the total_group and group_total variables to the dataset.
`group_diff(dat, project, group, sort_by, value, name, lag, create_group_ID, drop_total_col)` creates a group lagged difference variable within groups. `value`, a numeric variable, is first summed by the variable(s) in `group`, then the difference within-group is calculated. `sort_by` is a date variable to order `dat` by. The "group_total" variable gives the total value by group and can be dropped by setting `drop_total_col = TRUE`. A group ID column can be created using the variables in `group` by setting `create_group_ID = TRUE`.
`group_cumsum(dat, project, group, sort_by, value, name, create_group_ID, drop_total_col)` sums `value` by group, then cumulatively sums within `group`. For example, a running sum by trip can be made by entering variables that identify unique vessels and trips into `group` and a numeric variable (such as catch or number of hauls) into `value`. The group total variable can be dropped by setting `drop_total_col = TRUE`. A group ID column can be created using the variables in group by setting `create_group_ID = TRUE`.
`set_quants(dat, x, quant.cat = c(0.2, 0.25, 0.4), custom.quant, name)` transform `x` into a quantile category using pre-defined (`quant.cat`) or user-defined (`custom.quant`) quantiles.

Predefined quantiles 0.1 (0%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, 100%)
0.2 (0%, 20%, 40%, 60%, 80%, 100%)
0.25 (0%, 25%, 50%, 75%, 100%)
0.4 (0%, 10%, 50%, 90%, 100%)

5.0.1.3 Dummy

`dummy_num(dat, var, value, opts, name)` creates a binary variable contrasting a variable (`var`) based on a splitting value (`value`) and splitting rule (`opts`). The function is used to contrast before/after a policy is implemented or zone closures. `value` should be a year if `var` is a date variable, a factor level if `var` is a factor variable, a single or range of numbers if `var` is a numeric variable. `Opts` may be *"x_y"* or *"more_less"*. *"x_y"* sets each element of `var` to 1 if the element matches `value`, otherwise 0. For *"more_less"*, each element of `var` less than `value` is set to 0 and all elements greater than `value` are set to 1. If `var` is a factor, then elements that match `value` will be set to 1 and all other elements set to 0. Default is set to *"more_less"*.
`dummy_matrix(dat, x)` creates a dummy matrix of TRUE/FALSE based on values of variable `x` with dimensions *(number of observations in dataset)* (number of factors in x)* where each column is a unique factor level of `x`. Values are TRUE if the value in the column matches the column factor level and FALSE otherwise.

5.0.1.4 Nominal ID

`ID_var(dat, vars, name, type, drop)` is used to create a haul or trip identifier variable from on or more variables (`vars`). `type` is the class type of the new ID column. Choices are *"string"* and *"integer"*. *"string"* returns a character vector where each column in `vars` is combined and separated with an underscore `"_"`. *"integer"* returns an integer vector where each value corresponds to a unique group in `vars`. Set `drop = TRUE` to drop `vars` columns from `dat`.
`create_seasonal_ID(dat, seasonal.dat, use.location, use.geartype, sp.col, target)` uses a table of fishery season dates to create one or more fishery season identifier variables. The function matches fishery season dates provided in `seasonal.dat` to the first date variable in `dat`. If fishery season dates vary by location or gear type, set `use.location = TRUE` and `use.geartype = TRUE`. Output is a seasonID variable and/or multiple *seasonID*fishery* variables. The seasonID variable is a vector of fisheries. If a target fishery (`target`) is not defined, each row is the first observed fishery in `seasonal.dat` for which the fishery season dates encompasses date in `dat`. If target fishery is defined, then SeasonID is defined by whether the target fishery is open on the date in `dat` or a different fishery. The vector is filled with *"target"* or *"other"*. *SeasonID*fishery* variables are a TRUE/FALSE seasonID vector for each fishery (labeled by seasonID and

fishery) where TRUE indicates the dates for a given row in the main data table fall within the fishery dates for that fishery.

5.0.1.5 Spatial

`assignment_column(dat, project, gridfile, lon.dat, lat.dat, cat, closest.pt = FALSE, lon.grid = NULL, lat.grid = NULL, hull.polygon = FALSE, epsg = NULL)` assigns observations (hauls) to zones. `gridfile` is the spatial dataset defining zone boundaries, `lon.dat` and `lat.dat` are variables in `dat` used to assign each observation to a zone. `cat` is a variable or list in `gridfile` that identifies the individual areas or zones. `lon.grid` and `lat.grid`are properties or lists from `gridfile` containing longitude and latitude data. These data are required if `gridfile` is not class *sf*. `hull.polygon` and `closest.pt` are logical optional arguments. Set `hull.polygon = TRUE` if spatial data creating polygons are sparse or irregular. To assign observations that fall outside any zones to the closest zone polygon set `closest.pt = TRUE`. `epsg` is the spatial reference system. Specify `epsg` if the projected coordinate systems do not match.
`create_dist_between(dat, start, end, units, name)` adds a distance between two points variable to `dat`. `start` and `end` locations define how to find the lon/lat of starting and ending locations. Options are a port (specify port variable name), observed latitude and longitude location (specify variable name containing longitude data then latitude data as a character string), or the centroid of the observed fishery zone location (specify 'centroid'). Distance is returned in the requested `units` (*"miles", "kilometers", "midpoint"*). Additional arguments may be required, depending upon the `start` and `end` arguments. These are added through prompts.

Additional arguments *Port arguments*

portTable - Table contains the latitude and longitude of ports.

Centroids arguments
gridfile - Spatial data file.
lon.dat - Longitude variable from dat.
lat.dat - Latitude variable from dat.
lon.grid - Property or variable from gridfile containing longitude data.
lat.grid - Property or variable from gridfile containing latitude data.
cat - Property or variable from gridfile that identifies the individual areas or zones.

`create_mid_haul(dat, start=c("lon", "lat"), end=c("lon", "lat"), name)` returns the latitude and longitude of the midpoint location of each haul. Each row of `dat` must be a unique haul. Requires a `start` and `end` point for each observation. `start` and `end` must be specified in the order of longitude then latitude.

`create_startingloc(dat, gridfile, portTable, trip_id, haul_order, starting_port, lon.dat, lat.dat, cat, name, lon.grid, lat.grid)` creates a variable containing the zone/area location of a vessel when choice of where to fish next was made. This variable is required for the full information model with Dahl's correction (logit_correction). Unique trips in `dat` are identified with `trip_id`. `haul_order` identifies the order of hauls within `trip_id`. Generally, the first zone of a trip is the departure port and is identified with `starting_port`. `gridfile` and `portTable` are additional data tables containing information on zone boundaries and port locations, respectively. These two data tables along with `lon.dat`, `lat.dat`, `cat`, `lon.grid`, and `lat.grid` are called by the `assignment_column()` function to assign starting port locations and haul locations to zones. `lon.grid` and `lat.grid` are required if `gridfile` is a data frame or list.

5.0.1.6 Temporal

`temporal_mod(dat, x, define.format, name)` extracts a time unit from `x`, date variable in `dat`, in the desired format (`define.format`). The FishSET function `date_parser()` is first called to convert the date variable into recognized class if required. The `as.Date()` function from the base package is then called to extract the time in the desired format. `define.format` can be a predefined format (see below) or user-defined.

Predefined formats:

```
Value    as.Date format      Output
------------------------------------------------------------------
year    "%Y"                year
month   "%Y/%m"             year and month
day     "%Y/%m/%d"          year, month, and day
hour    "%Y/%m/%d %H"       year, month, day and hour
minute  "%Y/%m/%d %H:%M"    year, month, day, hour, and minute
```

`create_duration(dat, start, end, units=c("week", "day", "hour", "minute"), name)` returns the duration of time, in specified `units`, between two temporal variables (`start`, `end`) in `dat`. A duration of time variable is required for other functions, such as `cpue()`.

5.0.1.7 Trip-level `haul_to_trip(dat, project, fun.numeric, fun.time, tripID)` collapses `dat` from haul to trip level. Unique trips are identified by `tripID`. `tripID` may be one or more variables from `dat`. `fun.numeric` and `fun.time` define how multiple observations for a trip are collapsed. Options are *min*, *mean*, *max*, and *sum* for numeric variables and *min*, *mean*, and *max* for temporal variables. For variables that are not numeric or dates, the first observation is used.
`create_trip_distance(dat, PortTable, trip_id, starting_port, starting_haul = c("Lon", "Lat"), ending_haul = c("Lon", "Lat"), ending_port, haul_order, name, a = 6378137, f = 1/298.257223563)` sums distance across a trip based on starting and ending ports and hauls in between. Inputs are the trips, ports, and hauls from the primary dataset, and the latitude and longitude of ports from the `PortTable`. The ellipsoid arguments, `a` and `f`, can be changed if an ellipsoid other than WGS84 is appropriate. See the geosphere R package.
`create_trip_centroid(dat, lon, lat, weight.var, ...)` returns the centroid, in longitude and latitude, for each trip. Specify a weighting variable (`weight.var`) to calculate the weighted centroid. Additional arguments can be added that define unique trips. If no additional arguments are added, each row will be treated as a unique trip.

6 Modeling Functions

6.1 Modeling functions

Model development in FishSET requires preparing the data and defining model parameters. This section details the required and optional data, model choices, and model output evaluation functions.

Generate necessary data

```
create_startingloc()         Starting location variable.
create_alternative_choice() Define alternative fishing choices.
create_expections()          Define expected catch/revenue matrix.
```

Check data

```
check_model_data()           Check model data for possible issues with data.
```

Design model

```
make_model_design()         Make model design.
discretefish_subroutine()   Run model.
```

Assess model output

```
globalcheck_view()          Model error.
model_out_view()            Model output.
Metrics model_fit()         Model comparison.
select_model()              Select best model.
```

###Generate necessary data All models require catch data, data on the chosen fishing location (choice), and a distance matrix. Price data, location when choice of where to fish next, and expected catch matrices are required for some models. Additional optional data that interact with travel distance or vary by location may also be added. These data and the functions to generate them are outlined next.

Distance matrix
This function returns the matrix of distances between observed and alternative fishing choices (where they could have fished but did not). The distance matrix is saved in the model design file in the FishSET database and pulled when the model run function is called.

```
create_alternative_choice(dat, project, alt_var, occasion, griddedDat, dist.unit, min.haul, gridfile, ca
```

The distance matrix can be generated from the primary haul-level data (`dat`) or from gridded data (`griddedDat`) such as sea surface temperature. We describe generating the distance matrix from `dat` first.

From `dat`
Define how to find the longitude and latitude for the starting (`occasion`) and alternative (`alt_var`) locations. Choices are the centroid of the zone where the haul occurred (*"centroid"*), a port variable in `dat` (such as disembarked port), or paired latitude and longitude variables in `dat` such as haul starting location. Distance can be returned in *"miles"*, *"kilometers"*, or *"meters"* (`dist.unit`). Remove zones with insufficient data by setting `min.haul` to the minimum number of hauls required.

If `alt_var` or `occasion` are *"centroid"* then the centroid of zones must be obtained. If a centroid table exists in the FishSET database and a zone identifier variables is in `dat`, then only `gridfile` needs to be defined. `gridfile` should the name of the gridfile, in quotes. For example, `gridfile = "gridname"`. If centroids or zone assignments must be calculated then `gridfile`, `cat`, `lon.dat`, and `lat.dat` must be specified. `hull.polygon`, `closest.pt`, `weight.var` are optional `lon.grid` and `lat.grid` must be defined if `gridfile` is a data frame. See `assignment_column()` for more details on optional arguments.

If `occasion` is a port variable, a port table containing the latitude and longitude of ports is required. If a port table does not exist in the FishSET database run the `load_port()` function before running `create_alternative_choice`.

If `alt_var` or `occasion` are paired longitude and latitude variables from `dat` then `alt_var` or `occasion` should be specified as *`c(lon, lat)`*.
From `griddedDat`
Alternatively, the distance matrix can be generated from a gridded dataset, such as sea surface temperature. In this case, only `dat`, `project`, and `griddedDat` must be specified. Columns in the gridded data file must be individual zones and must match the zone identifier variable in `dat`. The gridded data can vary over a second dimension, such as date, but must match a variable in `dat`. We recommend saving the gridded data file to the FishSET database using the `load_grid()` function.

Expected catch matrix
This function returns the expected catch or revenue matrix for alternative fishing zones (zones they could have chosen but that the fisher did not in this choice opportunity). This matrix is required to run the conditional logit model that utilizes an average revenue specification.

```
create_expectations(dat, project, catch, price, defineGroup, temp.var, temporal, calc.method, lag.metho
```

The `create_expectations()` function requires a catch variable. To return expected revenue, include a price or value variable from `dat`. `price` is multiplied against catch to generate revenue. If revenue exists in `dat` and you wish to use this revenue instead of price, then `catch` must be a vector of 1 of length equal to `dat`.

Expected catch/revenue can be calculated across the entire dataset (`defineGroup = "fleet"`) or within groups and across the entire record of catch (`temp.var = NULL`) or can take variation in catch over time into account. To take temporal patterns in catch into account, first identify the temporal variable (`temp.var`) in `dat` to use. Next, select whether to use a sequential (`temporal = "sequential"`) or daily timeline (`temporal = "daily"`). Sequential timeline sorts data by date but does not take into account that days may be missing. Daily timeline adds in these dates with NA (missing value). With sequential timeline, a window size of seven means catch would be the average over the past seven fishing days. Whereas, with daily timeline, it would be the average over seven calendar days. `year.lag` can be 0 (current year), 1 (previous year's data), or any other viable years to go back. Window size (`temp.window`) is in days and can be current day (0), a week (7), or any other numeric value. `temp.lag` is numeric and in days. For each day x in `temp.var`, catch is the average catch across w days (`temp.window`) starting d days (`temp.lag`) and y years (`year.lag`) before x.

Using the specified moving window parameters, a matrix of average catch is created with *zone*group* creating rows and ***date*** the columns. This is the standard average catch (`calc.method = "standardAverage"`). Alternatively, you can use the simple lag regression of the mean (`calc.method = "simpleLag"`) which calculates the predicted value for each zone*group and date given regression coefficients p for each zone*group. The lag method (`lag.method`), can use regression over entire group (*"simple"*) or for grouped time periods (*"grouped"*).

The expected catch matrix is pulled from the matrix of calculated catches for each date and zone. The matrix is of dimensions *(number of rows of dat)*(number of alternatives)*. Expected catch is filled out by mapping the calculated catch for each zone given the observed date (if specified) and group (if specified) in `dat`. Note that empty catch values are considered to be times of no fishing activity and are not included. Values of 0 in the catch variable are considered times when fishing activity occurred but with no catch. These zero values are included in calculations. Sparsity in data should be considered when deciding how to take into account that catch may vary over time. Check for sparsity with the `temp_obs_table()` function. A broader window size or using the entire temporal record may be necessary. Empty catch values and empty expected catch values can be filled but only on a limited basis as doing so can lead to biased or misleading results. If there are a lot of empty values, consider changing the temporal arguments to reduce data sparsity. `empty.catch` can be *0*, the average of all catch values (*"allCatch"*") or the average of grouped catch values (*"groupedCatch"*). `empty.expectation` can be 0 or 1e-04.

The function returns four expected catch or expected revenue matrices based on predefined window size and lags in days and years and all other user-defined arguments :

```
selected temporal arguments
expected catch/revenue based the previous two days (short-term) catch,
expected catch/revenue based the previous seven days (medium-term) catch,
and expected catch/revenue based on the previous years (long-term) catch.
```

Starting location

The starting location vector is required for the full information model with Dahl's correction (`logit_correction`). The vector is the zone location of a vessel when the decision of where to fish next was made. Generally, the first zone of a trip is the departure port. Create the starting location vector before creating the model design file (`make_model_design()`).

```
create_startingloc(dat, gridfile, portTable, trip_id, haul_order, starting_port, lon.dat, lat.dat, cat,
```

The `create_startingloc()` function adds a starting location vector to the dataset (`dat`).

A zonal identifier variable is required. If zone identifier variable does not exist, the `assignment_column()` function is called to assign starting port locations and haul locations to zones and the following arguments are required: `gridfile`, `lon.dat`, `lat.dat`, `cat`, `lon.grid`, `lat.grid`. See `assignment_column` documentation for details.

The first zone is generally the zone of the departure port. The table containing port latitude and longitudes (`portTable`) is required. This should be in the FishSET database. Also required is the variable in `dat` containing names of starting ports (`starting_port`). `trip_id` is a variable in `dat` containing unique trip identifiers. `haul_order` is a variable in `dat` containing the order of hauls within trips.

Additional optional data
`vars1` is a character string of additional *travel-distance* variables to include in the model, such as vessel characteristics. `vars2` is a character string of additional variables to include in the model, wind speed within zones. These depend on the likelihood. The details of the functions depend on the likelihood function and are outlined in the FishSET Help Manual and likelihood function documentation.

###Model choices Model choices include the likelihood functions, optimization method, optimization options, and starting parameter values. These, along with the required and optional data are specified in the `model_design_function()`. The model design file is saved to the FishSET database and called by the `discretefish_subroutine()` function to run the models.

```
make_model_design(dat, project, catchID, replace = TRUE, likelihood = NULL, initparams, optimOpt, metho

discretefish_subroutine(project, select.model = FALSE)
```

The model design function requires the variable name in `dat` containing catch data. The distance and expected catch/revenue matrices do not have to be specified. They will be pulled based on `project` from the FishSET database. `startloc` is the name of the variable containing starting locations, if required. `priceCol` is optional argument to specify the variable in `dat` containing price data. It is required for the expected profit models (*epm_normal, epm_weibull, epm_lognormal*). `vars1` is a character string of optional *travel-distance* variables and `vars2` is a character string of optional *grid-varying* variables. These two arguments should be *NULL* if no additional data is being included. `polyn` is the correction polynomial degree. It is required for the *logit_correction* model.

FishSET allows only one model design file per project. Multiple models can be added to this file. Models are added on at a time using the `make_model_design()` function. Identify individual models with `model.name`. Models are added to the model design file by setting `replace = FALSE`. `replace = TRUE` will remove the existing model design file and only the currently specified model in `make_model_design()` will be in the model design file.

The remaining arguments in `make_model_design()` are detailed below.

Likelihood functions (`likelihood`)

FishSET has six built-in likelihood functions.

Function	Likelihood name	Reference
logit_c	Conditional logit likelihood	McFadden 1974
logit_avgcat	Average catch multinomial logit procedure	Abbot and Wilen 2011
logit_correction	Full information model with Dahl's correction function	Dahl 2002
epm_normal	Expected profit model with normal catch function	Haynie and Layton 2010
epm_weibull	Expected profit model with Weibull catch function	Haynie and Layton 2010
epm_lognormal	Expected profit model with lognormal catch function	Haynie and Layton 2010

See function documentation or the Help Manual for more details.

Initial parameter values (`initparams`)

The number of initial parameter values and the order depend upon the specified likelihood function and the number of travel-distance and grid-varying variables included.

logit_c
Starting parameter values take the order of `c[(alternative-specific parameters), (travel-distance parameters)]`. The length is the *number of alternative-specific variables* plus *number of travel-distance variables.*

logit_avgcat
Starting parameter values take the order of: `c[(average-catch parameters), (travel-distance parameters)]`. The length is the *(number of average-catch variables)*(k-1)* plus the *number of travel-distance variables.* `k` equals the number of alternative fishing choices.

logit_correction
starting parameter order takes: `c[(marginal utility from catch), (catch-function parameters), (polynomial starting parameters), (travel-distance parameters), (catch sigma)]`. The marginal utility from catch and catch sigma are of length equal to unity respectively. The catch-function and travel-distance parameters are of length *(number of catch variables)*(k)* and *number of cost variables* respectively. The number of polynomial interaction terms is currently set to 2, so given the chosen degree `polyn` there should be *(((polyn+1)*2)+2)*(k)* polynomial starting parameters, where `k` equals the number of alternative fishing choices.

epm_normal
Starting parameters values take the order of: `c[(catch-function parameters), (travel-distance parameters), (catch sigma(s)), (scale parameter)]`. The catch-function and travel-distance parameters are of length *(number of catch-function variables)*(k)* and *(number of travel-distance variables)* respectively, where `k` equals the number of alternative fishing choices. The catch sigma(s) are either of length equal to unity or length `k` if estimating location-specific catch sigma parameters. The scale parameter is of length equal to unity.

epm_weibull
Starting parameter values takes the order of: `c[(catch-function parameters), (travel-distance parameters), (catch sigma(s)), (scale parameter)]`. The catch-function and travel-distance parameters are of length *(number of catch-function variables)*(k)* and *(number of travel-distance variables)* respectively, where `k` equals the number of alternative fishing choices. The catch sigma(s) are either of length equal to unity or length `k` if estimating location-specific catch sigma parameters. The scale parameter is of length equal to unity.

epm_lognormal
Starting parameter values takes the order of: `c[(catch-function parameters), (travel-distance parameters), (catch sigma(s)), (scale parameter)]`. The catch-function and travel-distance parameters are of length *(number of catch-function variables)*(k)* and *(number of travel-distance variables)* respectively, where `k` equals the number of alternative fishing choices. The catch sigma(s) are either of length equal to unity or length `k` if estimating location-specific catch sigma parameters. The scale parameter is of length equal to unity.

Optimization options (`optimOpt`) Maximum number of iterations
Relative convergence tolerance
Report frequency
Level of tracing information returned

As a guide, the optimization function `optim` in the **stats** package defaults to 100 for derivative-based methods, 500 for "Nelder-Mead" and 10000 for "SANN". The relative tolerance should be very small. The default is 1e-08. Report frequency is the frequency of reports such as every 10 iterations for "BFGS" or every 100 temperatures for "SANN". Tracing must be 1 or higher for reports to be returned. Higher values return more details.

Optimization method (`methodname`)
Choices are "BFGS", "L-BFGS-B", "Nelder-Mead", "Brent", "CG", and "SANN". Default is "BFGS".

Running and saving models To run the defined models, use `discretefish_subroutine(project, select.model = FALSE)`. Each model can take 10 or more minutes to run.

It may be desirable to identify the "best" or preferred models for future reference. Identifying the "best" model can be done in the `discretefish_subroutine()` function by setting `select.model = TRUE` or using the `select_model(project, overwrite_table=FALSE)` function. Both produce a *modelChosen* table that is saved to the FishSET database. This table is not used in any functions. R Console

In the R console, an interactive data table can be opened by setting select.model in discretefish_subroutine() to TRUE or with the select_model() function. `select_model()` The interactive table contains the name of the model and model measures of fit for each model. Users can delete models from the table and select the preferred model by checking the selected box. The table is saved to the FishSET database with two new columns added, a TRUE/FALSE selected column and the date it was selected if overwrite_table is TRUE. The table is saved with the phrase 'modelChosen' in the FishSET database. FishSET GUI Identifying and recording the "best" or preferred model is done by checking the Selected box next to the desired model in the Measures of Fit table in the Model Comparison subtab. Click the blue Save Table button to record this selection. Use the Delete Row button to remove a model.

###Model output evaluation Output from the model is saved to the FishSET database in three tables. These tables are saved based on the project name, table phrase, and the date the model was run in YYMD format. The model output tables all contain the phrase *modelout* and the error output tables all contain *ldglobalcheck*.

The first table contains the initial global log likelihood, initial log likelihood for alternative choices, and starting parameters. Use these for assessing the cause of model errors. The second table contains information on the model output, such as convergence, standard errors, and the inverse Hessian matrix. The model comparison or measure of fit table contains AIC, AIC_c, BIC, Pseudo R^2.

7 Policy and Welfare Analysis

8 Reporting

FishSET includes a report template. It is written in RMarkdown and allows for output from FishSET to be easily incorporated into a shareable report. The template is divided into sections, each with a series of guiding questions and suggestions for your analysis. These questions will prompt you to explain how important aspects of your analysis were conducted, such as how the data was prepared, how models were defined, and which plots and tables to include. This is a suggested report structure. Sections and subsections can be added, deleted, or moved.

The report template is called `report_tempate.Rmd` and is available in the *report* folder of the FishSET package. Open the file and choose *Save as...* in the **File** tab in RStudio. Save the template to a meaningful name. This can be done for each project. The template contains guidance on working in RMarkdown and examples of script.