In [1]:

```
#%notebook "G:\Gantner Lab\Katy\K-soft\Jupyter Saving File.ipynb"

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import seaborn as sns
import statistics
from string import ascii_letters
import statannotations
from statannotations.Annotator import Annotator
from scipy import stats
from scipy.stats import stats, mannwhitneyu as mwu, normaltest
from autograd.scipy.special import expit, logit
import utils
import os
import itertools
# from utils import *
# import PIL
# from survive import datasets
# from survive import SurvivalData
# from survive import KaplanMeier, Breslow, NelsonAalen
# from sklearn.impute import SimpleImputer
# from lifelines import KaplanMeierFitter, CoxPHFitter, NelsonAalenFitter
# from lifelines.statistics import logrank_test
# from lifelines.utils import median_survival_times
# from lifelines.plotting import plot_lifetimes, add_at_risk_counts, plot_interval_censored_lifetim
# from lifelines.statistics import pairwise_logrank_test
# import kaplanmeier as km

plt.style.use('bmh')

#This line will show me the outputs inline in the notebook
%matplotlib inline

# Setting options to make data frame readable inline
pd.set_option('display.max_rows', 10)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', 1000)
pd.set_option('display.colheader_justify', 'center')
pd.set_option('display.precision', 3)
```

In [2]:

```
# #### Revision 9/27  ###

# ## use the path where the files are

path = r"G:\Gantner Lab\Katy\K-soft\2022_09_Kp Data CSVs - TEST"
bpath,dirname = os.path.split(path)
# neighborpath = os.path.join(bpath, dirname.replace("_ROIanalysis",""))


## Write a method to get all of the files in a given folder that end with .csv
def get_multiple_csv(folder, ext):
        return [os.path.join(folder,x) for x in os.listdir(folder) if x.endswith(ext)]
csvlist = get_multiple_csv(path, ".csv")

# Show me the files you found


display(csvlist)
```

```
['G:\\Gantner Lab\\Katy\\K-soft\\2022_09_Kp Data CSVs - TEST\\2022_0927_CX3CR1_tdTom_Kp_Vitals_Tpyx
l_7day.csv']
```

In [3]:

```
## Ok - 9/27/22 ##

def combine_csv(df):
#Combine the csv files into one big ass dataframe
    li = []

    for filename in df:

        df = pd.read_csv(filename, index_col=0, header=0)

        li.append(df)

        frame = pd.concat(li, axis=0, ignore_index=False)
        frame.head

#renaming as df for ease
        df = frame
        df.head
#     #Setting options to make data frame readable inline
#     pd.set_option('display.max_rows', 10)
#     pd.set_option('display.max_columns', None)
#     pd.set_option('display.width', 1000)
#     pd.set_option('display.colheader_justify', 'center')
#     pd.set_option('display.precision', 3)
#     df.fillna('', inplace=True)

#     # This splits the name into 4 columns; by default splitting is done on the basis of single sp
#     df1 = (df.Label.str.split(expand=True))
#     # This renames the new columns
#     df1.rename(columns={0:'Well', 1:'Channel', 2:'Frame_#', 3:'ROI_ID'},inplace=True)

# #   This drops the unused columns from the original dataframe
#     df = df.drop(columns=['Label'])

# #   This concatenates the two dataframes axis 1 - columns axis 0 - rows
#     ndf = pd.concat([df1,df], axis=1)
#     ndf["Condition"] = ndf.insert(0, "Condition", "")
    return(df)


tdf = combine_csv(csvlist)
# tdf.dropna(axis=0, how="all")

# ndf1 = tdf.copy

display(tdf)



# working data frame is now tdf
```

Experiment	Type	Genotype	Act_Dose	ID	Extype	Sex	GT	Dosegrp	uqID	Hour	Day	BM	SpO2	H
1.0	S	cKO	1000.0	M129	0.0	1	2.0	1	129.0	Pre	Pre	23.27	98.38	429.0
1.0	S	cKO	1000.0	M130	0.0	1	2.0	1	130.0	Pre	Pre	23.33	97.75	558.4
1.0	S	WT	1000.0	M131	0.0	1	1.0	1	131.0	Pre	Pre	23.69	96.14	447.4
1.0	S	WT	1000.0	M132	0.0	1	1.0	1	132.0	Pre	Pre	23.14	97.29	391.4
2.0	S	WT	1000.0	M197	0.0	1	1.0	1	197.0	Pre	Pre	25.63	98.00	395.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

	Type	Genotype	Act_Dose	ID	Extype	Sex	GT	Dosegrp	uqID	Hour	Day	BM	SpO2	H
Experiment														
NaN	NaN	NaN	NaN	NaN	NaN	FALSE	NaN	1	NaN	NaN	NaN	NaN	NaN	Na
NaN	NaN	NaN	NaN	NaN	NaN	FALSE	NaN	1	NaN	NaN	NaN	NaN	NaN	Na
NaN	NaN	NaN	NaN	NaN	NaN	FALSE	NaN	1	NaN	NaN	NaN	NaN	NaN	Na
NaN	NaN	NaN	NaN	NaN	NaN	FALSE	NaN	1	NaN	NaN	NaN	NaN	NaN	Na
NaN	NaN	NaN	NaN	NaN	NaN	FALSE	NaN	1	NaN	NaN	NaN	NaN	NaN	Na

1696 rows × 31 columns

In [4]:

```
tdf.dropna(axis=0, how="all")
sns.set(style="white")

ndf = tdf.copy()
cols = ["Genotype","Day","BM", "SpO2", "HR"]
physdf = pd.DataFrame(ndf, columns=cols)

physdf.dropna()
```

Out[4]:

	Genotype	Day	BM	SpO2	HR
Experiment					
1.0	cKO	Pre	23.27	98.38	429.00
1.0	cKO	Pre	23.33	97.75	558.44
1.0	WT	Pre	23.69	96.14	447.43
1.0	WT	Pre	23.14	97.29	391.43
2.0	WT	Pre	25.63	98.00	395.25
...	...	...	...	...	...
10.0	cKO	7	20.89	89.33	409.33
10.0	cKO	7	23.77	98.25	358.50
10.0	cKO	7	28.83	46.33	427.00
10.0	WT	7	25.25	43.50	340.50
11.0	cKO	7	18.64	74.00	555.00

369 rows × 5 columns

In [5]:

```
WT_BM_0 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="Pre"),"BM"].values
WT_BM_0 = WT_BM_0[~np.isnan(WT_BM_0)]

WT_BM_1 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="1"),"BM"].values
WT_BM_1 = WT_BM_1[~np.isnan(WT_BM_1)]

WT_BM_2 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="2"),"BM"].values
WT_BM_2 = WT_BM_2[~np.isnan(WT_BM_2)]

WT_BM_3 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="3"),"BM"].values
WT_BM_3 = WT_BM_3[~np.isnan(WT_BM_3)]

WT_BM_4 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="4"),"BM"].values
```

```
WT_BM_4 = WT_BM_4[~np.isnan(WT_BM_4)]

WT_BM_5 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="5"),"BM"].values
WT_BM_5 = WT_BM_5[~np.isnan(WT_BM_5)]

WT_BM_6 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="6"),"BM"].values
WT_BM_6 = WT_BM_6[~np.isnan(WT_BM_6)]

WT_BM_7 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="7"),"BM"].values
WT_BM_7 = WT_BM_7[~np.isnan(WT_BM_7)]

print(WT_BM_4)
```

```
[23.2  21.13 25.93 20.62 28.89 27.64 27.88 19.48 25.93 23.76 22.7  23.25
 20.44 23.42 23.45 22.2  26.24 25.31 19.36 24.06 22.01 25.13 24.82 25.17
 25.15 22.59 25.11 24.06]
```

In [25]:

```
# This section will put the Mass stats onto the seaborn graphs using statannotator
# https://github.com/trevismd/statannotations-tutorials/blob/main/Tutorial_1/Statannotations-Tutori


# # Method which makes a boxplot and a stripplot, adds stats and saves figure

# def Mass_plot_stat_save(path=r"G:\Gantner Lab\Katy\K-soft\2022_09_Kp Data CSVs - TEST\Saving Fold
##  This separates out dataframes from "physdf" by genotype and timepoint (Day) and gets the values

WT_BM_0 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="Pre"),"BM"].values
WT_BM_0 = WT_BM_0[~np.isnan(WT_BM_0)]

WT_BM_1 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="1"),"BM"].values
WT_BM_1 = WT_BM_1[~np.isnan(WT_BM_1)]

WT_BM_2 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="2"),"BM"].values
WT_BM_2 = WT_BM_2[~np.isnan(WT_BM_2)]

WT_BM_3 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="3"),"BM"].values
WT_BM_3 = WT_BM_3[~np.isnan(WT_BM_3)]

WT_BM_4 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="4"),"BM"].values
WT_BM_4 = WT_BM_4[~np.isnan(WT_BM_4)]

WT_BM_5 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="5"),"BM"].values
WT_BM_5 = WT_BM_5[~np.isnan(WT_BM_5)]

WT_BM_6 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="6"),"BM"].values
WT_BM_6 = WT_BM_6[~np.isnan(WT_BM_6)]

WT_BM_7 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="7"),"BM"].values
WT_BM_7 = WT_BM_7[~np.isnan(WT_BM_7)]

cKO_BM_0 = physdf.loc[(physdf["Genotype"]=="cKO") & (physdf["Day"]=="Pre"),"BM"].values
cKO_BM_0 = cKO_BM_0[~np.isnan(cKO_BM_0)]

cKO_BM_1 = physdf.loc[(physdf["Genotype"]=="cKO") & (physdf["Day"]=="1"),"BM"].values
cKO_BM_1 = cKO_BM_1[~np.isnan(cKO_BM_1)]

cKO_BM_2 = physdf.loc[(physdf["Genotype"]=="cKO") & (physdf["Day"]=="2"),"BM"].values
cKO_BM_2 = cKO_BM_2[~np.isnan(cKO_BM_2)]

cKO_BM_3 = physdf.loc[(physdf["Genotype"]=="cKO") & (physdf["Day"]=="3"),"BM"].values
cKO_BM_3 = cKO_BM_3[~np.isnan(cKO_BM_3)]

cKO_BM_4 = physdf.loc[(physdf["Genotype"]=="cKO") & (physdf["Day"]=="4"),"BM"].values
cKO_BM_4 = cKO_BM_4[~np.isnan(cKO_BM_4)]

cKO_BM_5 = physdf.loc[(physdf["Genotype"]=="cKO") & (physdf["Day"]=="5"),"BM"].values
```

```
cKO_BM_5 = cKO_BM_5[~np.isnan(cKO_BM_5)]

cKO_BM_6 = physdf.loc[(physdf["Genotype"]=="cKO") & (physdf["Day"]=="6"),"BM"].values
cKO_BM_6 = cKO_BM_6[~np.isnan(cKO_BM_6)]

cKO_BM_7 = physdf.loc[(physdf["Genotype"]=="cKO") & (physdf["Day"]=="7"),"BM"].values
cKO_BM_7 = cKO_BM_7[~np.isnan(cKO_BM_7)]


## This takes the separated out dataframes from above and compares them using a Mann-Whitney U test
Mass_results = [
#                     mwu(WT_BM_0, WT_BM_1, alternative="two-sided"),
#                     mwu(WT_BM_0, WT_BM_2, alternative="two-sided"),
#                     mwu(WT_BM_0, WT_BM_3, alternative="two-sided"),
#                     mwu(WT_BM_0, WT_BM_4, alternative="two-sided"),
#                     mwu(WT_BM_0, WT_BM_5, alternative="two-sided"),
#                     mwu(WT_BM_0, WT_BM_6, alternative="two-sided"),
#                     mwu(WT_BM_0, WT_BM_7, alternative="two-sided"),
#                     mwu(cKO_BM_0, cKO_BM_1, alternative="two-sided"),
#                     mwu(cKO_BM_0, cKO_BM_2, alternative="two-sided"),
#                     mwu(cKO_BM_0, cKO_BM_3, alternative="two-sided"),
#                     mwu(cKO_BM_0, cKO_BM_4, alternative="two-sided"),
#                     mwu(cKO_BM_0, cKO_BM_5, alternative="two-sided"),
#                     mwu(cKO_BM_0, cKO_BM_6, alternative="two-sided"),
#                     mwu(cKO_BM_0, cKO_BM_7, alternative="two-sided"),

                mwu(WT_BM_0, cKO_BM_0, alternative="two-sided"),
                mwu(WT_BM_1, cKO_BM_1, alternative="two-sided"),
                mwu(WT_BM_2, cKO_BM_2, alternative="two-sided"),
                mwu(WT_BM_3, cKO_BM_3, alternative="two-sided"),
                mwu(WT_BM_4, cKO_BM_4, alternative="two-sided"),
                mwu(WT_BM_5, cKO_BM_5, alternative="two-sided"),
                mwu(WT_BM_6, cKO_BM_6, alternative="two-sided"),
                mwu(WT_BM_7, cKO_BM_7, alternative="two-sided"),
                ]
# ## This prints the results ##
print("Mass Stats: \n", Mass_results[0], "\n")
# print("WT Pre vs WT 24-Hr", Mass_results[0])
# print("cKO Pre vs cKO 24-Hr", Mass_results[1])
# print("WT Pre vs cKO Pre", Mass_results[2])
# print("WT 24-Hr vs cKO 24-Hr", Mass_results[3])

M_pvalues = [result.pvalue for result in Mass_results]
# print(M_pvalues)
# print("Mass Stats", Mass_results[3])

# Setting seaborn aesthetics
# No gridlines, white background
sns.set(style="white")

# Create an array with the colors you want
colors2 = ["#16A085", "#76448A"]
color1 = ["white", "white"]

#Set the palette
sns.set_palette(sns.color_palette(colors2))

# # Putting the parameters in a dictionary avoids code duplication
# # since we use the same for `sns.boxplot` and `Annotator` calls

plotting_parameters = {
'data':    physdf,
'x':       'Day',
'y':       'BM',
'hue':'Genotype',
'hue_order':['WT', 'cKO'],
```

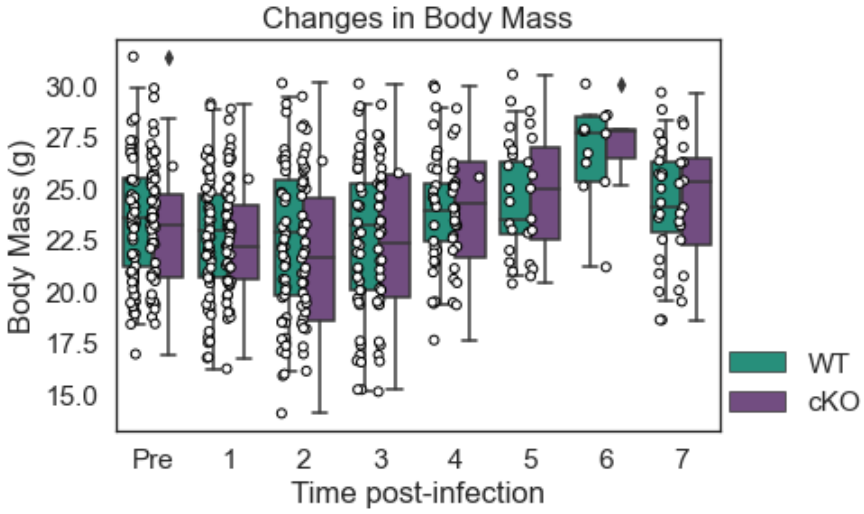
```
'order':['Pre','1', '2', '3', '4', '5', '6', '7'],
'palette': colors2,
}

pairs = [
#     (("Pre", "WT"), ("1", "WT")),
#     (("Pre", "WT"), ("2", "WT")),
#     (("Pre", "WT"), ("3", "WT")),
#     (("Pre", "WT"), ("4", "WT")),
#     (("Pre", "WT"), ("5", "WT")),
#     (("Pre", "WT"), ("6", "WT")),
#     (("Pre", "WT"), ("7", "WT")),

#     (("Pre", "cKO"), ("1", "cKO")),
#     (("Pre", "cKO"), ("2", "cKO")),
#     (("Pre", "cKO"), ("3", "cKO")),
#     (("Pre", "cKO"), ("4", "cKO")),
#     (("Pre", "cKO"), ("5", "cKO")),
#     (("Pre", "cKO"), ("6", "cKO")),
#     (("Pre", "cKO"), ("7", "cKO")),

    (("Pre", "WT"), ("Pre", "cKO")),
    (("1", "WT"), ("1", "cKO")),
    (("2", "WT"), ("2", "cKO")),
    (("3", "WT"), ("3", "cKO")),
    (("4", "WT"), ("4", "cKO")),
    (("5", "WT"), ("5", "cKO")),
    (("6", "WT"), ("6", "cKO")),
    (("7", "WT"), ("7", "cKO")),

]

## Transform the p-values into scientific notation
formatted_M_pvalues = [f"p={p:.2e}" for p in M_pvalues]

#Make a boxplot separated by Genotype and a stripplot with individual data points
with sns.plotting_context('notebook', font_scale=1.4):
    ax = sns.boxplot(x="Day", y="BM", hue="Genotype", hue_order=["WT","cKO"], data=tdf, palette=col
    sns.stripplot(x="Day", y="BM", hue="Genotype", data=tdf, palette=color1, edgecolor="black", doc

# This sets up the legend and labels for the graph
    handles, labels = ax.get_legend_handles_labels()
#         plt.figure(figsize=(1,2),dpi=1200)
    plt.legend(handles[:2], labels[:2], bbox_to_anchor=(1.3,.302), loc="best", borderaxespad=1, fra
    plt.xlabel('Time post-infection')
    plt.ylabel('Body Mass (g)')
    plt.title("Changes in Body Mass")
#         plt.tight_layout()

# Add Statistical annotations to the plot
    annotator = Annotator(ax, pairs, **plotting_parameters)
#     if not annotator.flags(["WRITEABLE"]):
#         annotator = annotator.copy(annotator)
    annotator.set_custom_annotations(formatted_M_pvalues)
    annotator.configure(test='Mann-Whitney', correction_format='replace', text_format="star", verbo
    _, results = annotator.apply_and_annotate()
#     annotator.apply_test()
    annotator.annotate()

# Show the Plot
    plt.show()

# # Save image
#     savepath=r"G:\Gantner Lab\Katy\K-soft\2022_09_Kp Data CSVs - TEST\Saving Folder"
#     tag = 0
#     while True:
```

```
#         fpath = os.path.join(savepath, "BodyMass_%3.3d.tif"%(tag))
#         if not os.path.exists(fpath) : break
#         tag += 1
#     fig = ax.get_figure()
# #       fig = ax.figure(figsize=(...))
#     fig.savefig(fpath, dpi=1200)


# Mass_plot_stat_save()
# print()
```

```
Mass Stats:
 MannwhitneyuResult(statistic=1734.0, pvalue=0.33396204858442824)

C:\Users\klafond\Anaconda3\lib\site-packages\statannotations\_Plotter.py:337: UserWarning: Invalid
x-position found. Are the same parameters passed to seaborn and statannotations calls? or are there
few data points?
  warnings.warn(
C:\Users\klafond\Anaconda3\lib\site-packages\statannotations\_Plotter.py:337: UserWarning: Invalid
x-position found. Are the same parameters passed to seaborn and statannotations calls? or are there
few data points?
  warnings.warn(
C:\Users\klafond\Anaconda3\lib\site-packages\statannotations\_Plotter.py:337: UserWarning: Invalid
x-position found. Are the same parameters passed to seaborn and statannotations calls? or are there
few data points?
  warnings.warn(
C:\Users\klafond\Anaconda3\lib\site-packages\statannotations\_Plotter.py:337: UserWarning: Invalid
x-position found. Are the same parameters passed to seaborn and statannotations calls? or are there
few data points?
  warnings.warn(
C:\Users\klafond\Anaconda3\lib\site-packages\statannotations\_Plotter.py:337: UserWarning: Invalid
x-position found. Are the same parameters passed to seaborn and statannotations calls? or are there
few data points?
  warnings.warn(
C:\Users\klafond\Anaconda3\lib\site-packages\statannotations\_Plotter.py:337: UserWarning: Invalid
x-position found. Are the same parameters passed to seaborn and statannotations calls? or are there
few data points?
  warnings.warn(
C:\Users\klafond\Anaconda3\lib\site-packages\statannotations\_Plotter.py:337: UserWarning: Invalid
x-position found. Are the same parameters passed to seaborn and statannotations calls? or are there
few data points?
  warnings.warn(
C:\Users\klafond\Anaconda3\lib\site-packages\statannotations\_Plotter.py:337: UserWarning: Invalid
x-position found. Are the same parameters passed to seaborn and statannotations calls? or are there
few data points?
  warnings.warn(
C:\Users\klafond\Anaconda3\lib\site-packages\statannotations\_Plotter.py:337: UserWarning: Invalid
x-position found. Are the same parameters passed to seaborn and statannotations calls? or are there
few data points?
  warnings.warn(
C:\Users\klafond\Anaconda3\lib\site-packages\statannotations\_Plotter.py:337: UserWarning: Invalid
x-position found. Are the same parameters passed to seaborn and statannotations calls? or are there
few data points?
  warnings.warn(
C:\Users\klafond\Anaconda3\lib\site-packages\statannotations\_Plotter.py:337: UserWarning: Invalid
x-position found. Are the same parameters passed to seaborn and statannotations calls? or are there
few data points?
  warnings.warn(
C:\Users\klafond\Anaconda3\lib\site-packages\statannotations\_Plotter.py:337: UserWarning: Invalid
x-position found. Are the same parameters passed to seaborn and statannotations calls? or are there
few data points?
  warnings.warn(
C:\Users\klafond\Anaconda3\lib\site-packages\statannotations\_Plotter.py:337: UserWarning: Invalid
x-position found. Are the same parameters passed to seaborn and statannotations calls? or are there
few data points?
  warnings.warn(
```

```
C:\Users\klafond\Anaconda3\lib\site-packages\statannotations\_Plotter.py:337: UserWarning: Invalid
x-position found. Are the same parameters passed to seaborn and statannotations calls? or are there
few data points?
  warnings.warn(
C:\Users\klafond\Anaconda3\lib\site-packages\statannotations\_Plotter.py:337: UserWarning: Invalid
x-position found. Are the same parameters passed to seaborn and statannotations calls? or are there
few data points?
  warnings.warn(
C:\Users\klafond\Anaconda3\lib\site-packages\statannotations\_Plotter.py:337: UserWarning: Invalid
x-position found. Are the same parameters passed to seaborn and statannotations calls? or are there
few data points?
  warnings.warn(
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_23860/3114087147.py in <module>
    164
    165 # Add Statistical annotations to the plot
--> 166     annotator = Annotator(ax, pairs, **plotting_parameters)
    167 #     if not annotator.flags(["WRITEABLE"]):
    168 #         annotator = annotator.copy(annotator)

~\Anaconda3\lib\site-packages\statannotations\Annotator.py in __init__(self, ax, pairs, plot, data,
x, y, hue, order, hue_order, engine, verbose, **plot_params)
    104             self._plotter = None
    105         else:
--> 106             self._plotter = self._get_plotter(engine, ax, pairs, plot, data,
    107                                               x, y, hue, order, hue_order,
    108                                               verbose=verbose, **plot_params)

~\Anaconda3\lib\site-packages\statannotations\Annotator.py in _get_plotter(engine, *args, **kwargs)
    776         if engine_plotter is None:
    777             raise NotImplementedError(f"{engine} engine not implemented.")
--> 778         return engine_plotter(*args, **kwargs)
    779
    780     def _get_xy_params_horizontal(self, group_coord_1, group_coord_2,

~\Anaconda3\lib\site-packages\statannotations\_Plotter.py in __init__(self, ax, pairs, plot, data,
 x, y, hue, order, hue_order, verbose, **plot_params)
     92                                                 self.group_names)
     93         self.reordering = None
---> 94         self.value_maxes = self._generate_value_maxes()
     95
     96         self.structs = self._get_structs()

~\Anaconda3\lib\site-packages\statannotations\_Plotter.py in _generate_value_maxes(self)
    213             for child in self.ax.get_children():
    214
--> 215                 group_name, value_pos = self._get_value_pos(child, data_to_ax)
    216
    217                 if (value_pos is not None

~\Anaconda3\lib\site-packages\statannotations\_Plotter.py in _get_value_pos(self, child, data_to_a
x)
    314         if (type(child) == PathCollection
    315                 and len(child.properties()['offsets'])):
--> 316             return self._get_value_pos_for_path_collection(
    317                 child, data_to_ax)
    318

~\Anaconda3\lib\site-packages\statannotations\_Plotter.py in _get_value_pos_for_path_collection(sel
f, child, data_to_ax)
    330
    331         value_max = child.properties()['offsets'][:, value_coord].max()
--> 332         group_pos = float(np.round(np.nanmean(
    333             child.properties()['offsets'][:, group_coord]), 1))
    334
```

```
<__array_function__ internals> in nanmean(*args, **kwargs)

~\Anaconda3\lib\site-packages\numpy\lib\nanfunctions.py in nanmean(a, axis, dtype, out, keepdims)
    948     cnt = np.sum(~mask, axis=axis, dtype=np.intp, keepdims=keepdims)
    949     tot = np.sum(arr, axis=axis, dtype=dtype, out=out, keepdims=keepdims)
--> 950     avg = _divide_by_count(tot, cnt, out=out)
    951
    952     isbad = (cnt == 0)

~\Anaconda3\lib\site-packages\numpy\lib\nanfunctions.py in _divide_by_count(a, b, out)
    210         if isinstance(a, np.ndarray):
    211             if out is None:
--> 212                 return np.divide(a, b, out=a, casting='unsafe')
    213             else:
    214                 return np.divide(a, b, out=out, casting='unsafe')

ValueError: output array is read-only
```



In [14]:

```
# This section will put the SpO2 stats onto the seaborn graphs using statannotator
# https://github.com/trevismd/statannotations-tutorials/blob/main/Tutorial_1/Statannotations-Tutori


# # Method which makes a boxplot and a stripplot, adds stats and saves figure

def SpO2_plot_stat_save(path=r"G:\Gantner Lab\Katy\K-soft\2022_09_Kp Data CSVs - TEST\Saving Folder
##  This separates out dataframes from "physdf" by genotype and timepoint (Day) and gets the values

    WT_Sp_0 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="Pre"),"SpO2"].values
    WT_Sp_1 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="24 hour"),"SpO2"].values
    cKO_Sp_0 = physdf.loc[(physdf["Genotype"]=="cKO") & (physdf["Day"]=="Pre"),"SpO2"].values
    cKO_Sp_1 = physdf.loc[(physdf["Genotype"]=="cKO") & (physdf["Day"]=="24 hour"),"SpO2"].values

    ## This takes the separated out dataframes from above and compares them using a Mann-Whitney U
    SpO2_results = [mwu(WT_Sp_0, WT_Sp_1, alternative="two-sided"),
                    mwu(cKO_Sp_0, cKO_Sp_1, alternative="two-sided"),
                    mwu(WT_Sp_0, cKO_Sp_0, alternative="two-sided"),
                    mwu(WT_Sp_1, cKO_Sp_1, alternative="two-sided"),
                    ]
#     ## This prints the results ##
#     print("WT Pre vs WT 24-Hr", SpO2_results[0])
#     print("cKO Pre vs cKO 24-Hr", SpO2_results[1])
#     print("WT Pre vs cKO Pre", SpO2_results[2])
#     print("WT 24-Hr vs cKO 24-Hr", SpO2_results[3])

    Sp_pvalues = [result.pvalue for result in SpO2_results]
```

```
# Setting seaborn aesthetics
    # No gridlines, white background
    sns.set(style="white")

    # Create an array with the colors you want
    colors2 = ["#16A085", "#76448A"]
    color1 = ["white", "white"]

    #Set the palette
    sns.set_palette(sns.color_palette(colors2))

    # Putting the parameters in a dictionary avoids code duplication
    # since we use the same for `sns.boxplot` and `Annotator` calls

    plotting_parameters = {
    'data':    physdf,
    'x':       'Day',
    'y':       'SpO2',
    'hue':'Genotype',
    'hue_order':['WT', 'cKO'],
    'order':['Pre','24 hour'],
    'palette': colors2,
    }

    pairs = [
        (("Pre", "WT"), ("24 hour", "WT")),
        (("Pre", "cKO"), ("24 hour", "cKO")),
        (("Pre", "WT"), ("Pre", "cKO")),
        (("24 hour", "WT"), ("24 hour", "cKO")),
    ]

    formatted_Sp_pvalues = [f"p={p:.2e}" for p in Sp_pvalues]

    # Make a boxplot separated by Genotype and a stripplot with individual data points
    with sns.plotting_context('notebook', font_scale=1.4):
        ax = sns.boxplot(x="Day", y="SpO2", hue="Genotype", data=tdf, palette=colors2)
        sns.stripplot(x="Day", y="SpO2", hue="Genotype", data=tdf, hue_order=["WT","cKO"], palette=

    # This sets up the legend and labels for the graph
        # This sets up the legend and labels for the graph
        handles, labels = ax.get_legend_handles_labels()
#         plt.figure(figsize=(1,2),dpi=1200)
        plt.legend(handles[:2], labels[:2], bbox_to_anchor=(1.28,.302), loc="best", borderaxespad=1
        plt.xlabel('Time post-infection')
        plt.ylabel('SpO2 %')
        plt.title("Changes in Oxygen Saturation")
#         plt.tight_layout()

        #Add Statistical annotations to the plot
        annotator = Annotator(ax, pairs, **plotting_parameters)
        annotator.set_custom_annotations(formatted_Sp_pvalues)
        annotator.configure(test='Mann-Whitney',correction_format='replace', text_format="star")
        annotator.apply_test()
        annotator.annotate()

       #Show the Plot
        plt.show()

        #Save image
        savepath=r"G:\Gantner Lab\Katy\K-soft\2022_09_Kp Data CSVs - TEST\Saving Folder"
        tag = 0
        while True:
            fpath = os.path.join(savepath, "SpO2_%3.3d.tif"%(tag))
            if not os.path.exists(fpath) : break
            tag += 1
        fig = ax.get_figure()
```

```
        fig.savefig(fpath, dpi=1200)

SpO2_plot_stat_save()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_23860/956389311.py in <module>
     96         fig.savefig(fpath, dpi=1200)
     97
---> 98 SpO2_plot_stat_save()

~\AppData\Local\Temp/ipykernel_23860/956389311.py in SpO2_plot_stat_save(path)
     14
     15     ## This takes the separated out dataframes from above and compares them using a Mann-Wh
itney U test ##
---> 16     SpO2_results = [mwu(WT_Sp_0, WT_Sp_1, alternative="two-sided"),
     17                     mwu(cKO_Sp_0, cKO_Sp_1, alternative="two-sided"),
     18                     mwu(WT_Sp_0, cKO_Sp_0, alternative="two-sided"),

~\Anaconda3\lib\site-packages\scipy\stats\_mannwhitneyu.py in mannwhitneyu(x, y, use_continuity, al
ternative, axis, method)
    389
    390     x, y, use_continuity, alternative, axis_int, method = (
--> 391         _mwu_input_validation(x, y, use_continuity, alternative, axis, method))
    392
    393     x, y, xy = _broadcast_concatenate(x, y, axis)

~\Anaconda3\lib\site-packages\scipy\stats\_mannwhitneyu.py in _mwu_input_validation(x, y, use_conti
nuity, alternative, axis, method)
    131     x, y = np.atleast_1d(x), np.atleast_1d(y)
    132     if np.isnan(x).any() or np.isnan(y).any():
--> 133         raise ValueError('`x` and `y` must not contain NaNs.')
    134     if np.size(x) == 0 or np.size(y) == 0:
    135         raise ValueError('`x` and `y` must be of nonzero size.')

ValueError: `x` and `y` must not contain NaNs.
```

In [1]:

```
# This section will put the Heart Rate stats onto the seaborn graphs using statannotator
# https://github.com/trevismd/statannotations-tutorials/blob/main/Tutorial_1/Statannotations-Tutori


# # Method which makes a boxplot and a stripplot, adds stats and saves figure

def HR_plot_stat_save(path=r"G:\Gantner Lab\Katy\K-soft\2022_09_Kp Data CSVs - TEST\Saving Folder"):
##  This separates out dataframes from "physdf" by genotype and timepoint (Day) and gets the values

    ## Heart Rate (HR)
    WT_HR_0 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="Pre"),"HR"].values
    WT_HR_1 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="24 hour"),"HR"].values
    cKO_HR_0 = physdf.loc[(physdf["Genotype"]=="cKO") & (physdf["Day"]=="Pre"),"HR"].values
    cKO_HR_1 = physdf.loc[(physdf["Genotype"]=="cKO") & (physdf["Day"]=="24 hour"),"HR"].values


    ## This takes the separated out dataframes from above and compares them using a Mann-Whitney U
    HR_results = [mwu(WT_HR_0, WT_HR_1, alternative="two-sided"),
                  mwu(cKO_HR_0, cKO_HR_1, alternative="two-sided"),
                  mwu(WT_HR_0, cKO_HR_0, alternative="two-sided"),
                  mwu(WT_HR_1, cKO_HR_1, alternative="two-sided"),
                  ]
#     ## This prints the results ##
#     print("WT Pre vs WT 24-Hr", HR_results[0])
#     print("cKO Pre vs cKO 24-Hr", HR_results[1])
#     print("WT Pre vs cKO Pre", HR_results[2])
#     print("WT 24-Hr vs cKO 24-Hr", HR_results[3])

    HR_pvalues = [result.pvalue for result in HR_results]
```

```
# Setting seaborn aesthetics
    # No gridlines, white background
    sns.set(style="white")

    # Create an array with the colors you want
    colors2 = ["#16A085", "#76448A"]
    color1 = ["white", "white"]

    #Set the palette
    sns.set_palette(sns.color_palette(colors2))

    # Putting the parameters in a dictionary avoids code duplication
    # since we use the same for `sns.boxplot` and `Annotator` calls

    plotting_parameters = {
    'data':    physdf,
    'x':       'Day',
    'y':       'HR',
    'hue':'Genotype',
    'hue_order':['WT', 'cKO'],
    'order':['Pre','24 hour'],
    'palette': colors2,
    }

    pairs = [
        (("Pre", "WT"), ("24 hour", "WT")),
        (("Pre", "cKO"), ("24 hour", "cKO")),
        (("Pre", "WT"), ("Pre", "cKO")),
        (("24 hour", "WT"), ("24 hour", "cKO")),
    ]

    formatted_HR_pvalues = [f"p={p:.2e}" for p in HR_pvalues]

    #Make a boxplot separated by Genotype and a stripplot with individual data points
    with sns.plotting_context('notebook', font_scale=1.4):
        ax = sns.boxplot(x="Day", y="HR", hue="Genotype", data=tdf, palette=colors2)
        sns.stripplot(x="Day", y="HR", hue="Genotype", data=tdf, hue_order=["WT","cKO"], palette=cc
    # This sets up the legend and labels for the graph
       # This sets up the legend and labels for the graph
        handles, labels = ax.get_legend_handles_labels()
#         plt.figure(figsize=(1,2),dpi=1200)
        plt.legend(handles[:2], labels[:2], bbox_to_anchor=(1.9,.302), loc="best", borderaxespad=1,
        plt.xlabel('Time post-infection')
        plt.ylabel('Heart Rate (bpm)')
        plt.title("Changes in Heart Rate")
#         plt.tight_layout()


    # Add Statistical annotations to the plot
        annotator = Annotator(ax, pairs, **plotting_parameters)
        annotator.set_custom_annotations(formatted_HR_pvalues)
        annotator.configure(test='Mann-Whitney', correction_format='replace', text_format="star")
        annotator.apply_test()
        annotator.annotate()

       #Show the Plot
        plt.show()

        #Save image
        savepath=r"G:\Gantner Lab\Katy\K-soft\2022_09_Kp Data CSVs - TEST\Saving Folder"
        tag = 0
        while True:
            fpath = os.path.join(savepath, "HR_%3.3d.tif"%(tag))
            if not os.path.exists(fpath) : break
```

```
            tag += 1
        fig = ax.get_figure()
        fig.savefig(fpath, dpi=1200)

HR_plot_stat_save()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_16764/1154160823.py in <module>
     98         fig.savefig(fpath, dpi=1200)
     99
--> 100 HR_plot_stat_save()

~\AppData\Local\Temp/ipykernel_16764/1154160823.py in HR_plot_stat_save(path)
      9
     10     ## Heart Rate (HR)
---> 11     WT_HR_0 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="Pre"),"HR"].values
     12     WT_HR_1 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="24 hour"),"HR"].valu
es
     13     cKO_HR_0 = physdf.loc[(physdf["Genotype"]=="cKO") & (physdf["Day"]=="Pre"),"HR"].values

NameError: name 'physdf' is not defined
```

In [ ]:

```
# tdf.dropna(axis=0, how="all")
# sns.set(style="white")

# ndf = tdf.copy()
# cols = ["Genotype","Day","Mass", "SpO2", "HR"]
# # Mass_cols = ["Genotype", "Day", "Mass"]
# # SpO2_cols = ["Genotype", "Day", "SpO2"]
# # HR_cols = ["Genotype", "Day", "HR"]

# physdf = pd.DataFrame(ndf, columns=cols).dropna(how="all")



# # # Method which makes a boxplot and a stripplot and saves figure
# def Mass_save_fig(plotter, path=r"G:\Gantner Lab\Katy\K-soft\2022_0929_Ph_301_CSV\2022_0929_Ph_30
#     # Create an array with the colors you want
#     colors2 = ["#16A085", "#76448A"]

#     #Set the palette
#     sns.set_palette(sns.color_palette(colors2))

#     #Make a boxplot separated by Genotype and a stripplot with individual data points
#     ax = sns.boxplot(x="Day", y="Mass", hue="Genotype", data=tdf, palette=colors2)
#     sns.stripplot(x="Day", y="Mass", hue="Genotype", data=tdf, color='white', dodge=True, jitter=
#     plt.legend(bbox_to_anchor=(1.22,.302), loc="lower right", borderaxespad=0)
#     plt.xlabel('Time post-infection')
#     plt.ylabel('Body Mass (g)')
#     plt.title("Baseline and 24-hours post-infection Body Mass")

#     tag = 0
#     while True:
#         fpath = os.path.join(path, "BodyMass_%3.3d.tif"%(tag))
#         if not os.path.exists(fpath) : break
#         tag += 1
#     fig = ax.get_figure()
#     fig.savefig(fpath, dpi=1200)

# Mass_save_fig(ax)
```

In [ ]:

```
# ndf = tdf.copy()
# cols = ["Genotype","Day","Mass", "SpO2", "HR"]
```

```
# # Mass_cols = ["Genotype", "Day", "Mass"]
# # SpO2_cols = ["Genotype", "Day", "SpO2"]
# # HR_cols = ["Genotype", "Day", "HR"]

# physdf = pd.DataFrame(ndf, columns=cols).dropna(how="all")


# # Massdf = pd.DataFrame(ndf, columns=Mass_cols).dropna(how="all")
# # SpO2df = pd.DataFrame(ndf, columns=SpO2_cols).dropna(how="all")
# # HRdf = pd.DataFrame(ndf, columns=HR_cols).dropna(how="all")
# # display(Massdf)
```

In [ ]:

```
# ##  This separates out dataframes from "physdf" by genotype and timepoint (Day) and gets the valu

# WT_BM_0 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="Pre"),"Mass"].values
# WT_BM_1 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="24 hour"),"Mass"].values
# cKO_BM_0 = physdf.loc[(physdf["Genotype"]=="cKO") & (physdf["Day"]=="Pre"),"Mass"].values
# cKO_BM_1 = physdf.loc[(physdf["Genotype"]=="cKO") & (physdf["Day"]=="24 hour"),"Mass"].values

# ## This takes the separated out dataframes from above and compares them using a Mann-Whitney U te
# Mass_results = [mwu(WT_BM_0, WT_BM_1, alternative="two-sided"),
#                 mwu(cKO_BM_0, cKO_BM_1, alternative="two-sided"),
#                 mwu(WT_BM_0, cKO_BM_0, alternative="two-sided"),
#                 mwu(WT_BM_1, cKO_BM_1, alternative="two-sided"),
#                 ]
# ## This prints the results ##
# print("WT Pre vs WT 24-Hr", Mass_results[0])
# print("cKO Pre vs cKO 24-Hr", Mass_results[1])
# print("WT Pre vs cKO Pre", Mass_results[2])
# print("WT 24-Hr vs cKO 24-Hr", Mass_results[3])

# M_pvalues = [result.pvalue for result in Mass_results]

# ## SpO2...

# WT_Sp_0 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="Pre"),"SpO2"].values
# WT_Sp_1 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="24 hour"),"SpO2"].values
# cKO_Sp_0 = physdf.loc[(physdf["Genotype"]=="cKO") & (physdf["Day"]=="Pre"),"SpO2"].values
# cKO_Sp_1 = physdf.loc[(physdf["Genotype"]=="cKO") & (physdf["Day"]=="24 hour"),"SpO2"].values

# ## This takes the separated out dataframes from above and compares them using a Mann-Whitney U te
# SpO2_results = [mwu(WT_Sp_0, WT_Sp_1, alternative="two-sided"),
#                 mwu(cKO_Sp_0, cKO_Sp_1, alternative="two-sided"),
#                 mwu(WT_Sp_0, cKO_Sp_0, alternative="two-sided"),
#                 mwu(WT_Sp_1, cKO_Sp_1, alternative="two-sided"),
#                 ]
# ## This prints the results ##
# print("WT Pre vs WT 24-Hr", SpO2_results[0])
# print("cKO Pre vs cKO 24-Hr", SpO2_results[1])
# print("WT Pre vs cKO Pre", SpO2_results[2])
# print("WT 24-Hr vs cKO 24-Hr", SpO2_results[3])

# Sp_pvalues = [result.pvalue for result in SpO2_results]

# ## Heart Rate (HR)
# WT_HR_0 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="Pre"),"HR"].values
# WT_HR_1 = physdf.loc[(physdf["Genotype"]=="WT") & (physdf["Day"]=="24 hour"),"HR"].values
# cKO_HR_0 = physdf.loc[(physdf["Genotype"]=="cKO") & (physdf["Day"]=="Pre"),"HR"].values
# cKO_HR_1 = physdf.loc[(physdf["Genotype"]=="cKO") & (physdf["Day"]=="24 hour"),"HR"].values


# ## This takes the separated out dataframes from above and compares them using a Mann-Whitney U te
# HR_results = [mwu(WT_HR_0, WT_HR_1, alternative="two-sided"),
#                 mwu(cKO_HR_0, cKO_HR_1, alternative="two-sided"),
#                 mwu(WT_HR_0, cKO_HR_0, alternative="two-sided"),
```

```
#                mwu(WT_HR_1, cKO_HR_1, alternative="two-sided"),
#                ]
# ## This prints the results ##
# print("WT Pre vs WT 24-Hr", HR_results[0])
# print("cKO Pre vs cKO 24-Hr", HR_results[1])
# print("WT Pre vs cKO Pre", HR_results[2])
# print("WT 24-Hr vs cKO 24-Hr", HR_results[3])

# HR_pvalues = [result.pvalue for result in HR_results]
```

In [ ]:

```
# formatted_M_pvalues = [f"p={p:.2e}" for p in M_pvalues]
# print(formatted_M_pvalues)
```

In [ ]:

```
# # This section will put the stats onto the seaborn graphs using statannotator
# # https://github.com/trevismd/statannotations-tutorials/blob/main/Tutorial_1/Statannotations-Tutc


# # # Method which makes a boxplot and a stripplot and saves figure
# # def Mass_plot_stat_save(plotter, path=r"G:\Gantner Lab\Katy\K-soft\2022_0929_Ph_301_CSV\2022_09

# sns.set(style="white")

# # Create an array with the colors you want
# colors2 = ["#16A085", "#76448A"]

# #Set the palette
# sns.set_palette(sns.color_palette(colors2))

# # Putting the parameters in a dictionary avoids code duplication
# # since we use the same for `sns.boxplot` and `Annotator` calls

# plotting_parameters = {
# 'data':    physdf,
# 'x':       'Day',
# 'y':       'Mass',
# 'hue':'Genotype',
# 'hue_order':['WT', 'cKO'],
# 'order':['Pre','24 hour'],
# 'palette': colors2,
# }

# pairs = [
#     (("Pre", "WT"), ("24 hour", "WT")),
#     (("Pre", "cKO"), ("24 hour", "cKO")),
#     (("Pre", "WT"), ("Pre", "cKO")),
#     (("24 hour", "WT"), ("24 hour", "cKO")),
# ]

# formatted_M_pvalues = [f"p={p:.2e}" for p in M_pvalues]

# #Make a boxplot separated by Genotype and a stripplot with individual data points
# with sns.plotting_context('notebook', font_scale=1.4):
#     ax = sns.boxplot(x="Day", y="Mass", hue="Genotype", data=tdf, palette=colors2)
#     sns.stripplot(x="Day", y="Mass", hue="Genotype", data=tdf, color='white', dodge=True, jitter=
#     plt.legend(bbox_to_anchor=(1.22,.302), loc="lower right", borderaxespad=0)
#     plt.xlabel('Time post-infection')
#     plt.ylabel('Body Mass (g)')
#     plt.title("Baseline and 24-hours post-infection Body Mass")

#     #Add Statistical annotations to the plot
#     annotator = Annotator(ax, pairs, **plotting_parameters)
#     annotator.set_custom_annotations(formatted_M_pvalues)
#     annotator.annotate()
```

```
#    #Show the Plot
#     plt.show()

#     #Save image
#     savepath=r"G:\Gantner Lab\Katy\K-soft\2022_0929_Ph_301_CSV\2022_0929_Ph_301_Saving Folder"
#     tag = 0
#     while True:
#         fpath = os.path.join(savepath, "BodyMass_%3.3d.tif"%(tag))
#         if not os.path.exists(fpath) : break
#         tag += 1
#     fig = ax.get_figure()
#     fig.savefig(fpath, dpi=1200)

# # Mass_plot_stat_save(ax)
```

In [ ]:

```
# # This section will put the stats onto the seaborn graphs using statannotator
# # https://github.com/trevismd/statannotations-tutorials/blob/main/Tutorial_1/Statannotations-Tuto

# # Putting the parameters in a dictionary avoids code duplication
# # since we use the same for `sns.boxplot` and `Annotator` calls
# plotting_parameters = {
#     'data':    rfs,
#     'x':       'Subcategory',
#     'y':       'Goal',
#     'order':   subcat_order,
#     'palette': subcat_palette,
# }

# BM_annotator = Annotator(ax, pairs, ...)  # With ... = all parameters passed to seaborn's plotter
```

In [ ]:

```
# tdf2 = tdf.copy()
# # display(tdf2)

# # # Filter rows of averaged data #
# avdf = tdf2[tdf2['uqID'].str.contains('Av')]
# avdf.fillna('', inplace=True)
# display(avdf)



# # Compute the correlation matrix
# corr = avdf.corr()

# # Generate a mask for the upper triangle
# mask = np.triu(np.ones_like(corr, dtype=bool))

# # Set up the matplotlib figure
# f, ax = plt.subplots(figsize=(11, 9))

# # Generate a custom diverging colormap
# cmap = sns.diverging_palette(230, 20, as_cmap=True)

# # Draw the heatmap with the mask and correct aspect ratio
# sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
#             square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

In [ ]:

```
# # display(avdf)
# ndf = avdf.copy()
# BM_time = ndf.columns[ndf.columns.str.contains("BM_")].fillna("")
# # GT = tdf2[tdf2['Type'].str.contains('WT','cKO')]
# # Mass_df = pd.concat(BM_time,GT)
# # BM_time.fillna(0)
```

```
# # display(BM_time)
# Massdf = ndf.query('Type=="WT","cKO" & BM_time')

# display(Massdf)
# #sns.relplot(data=avdf, x=BM_time, y=GT)
```

In [ ]:

```
# #Filter data frame (currently avdf - only the averaged data) to only get the body mass measuremen
# avdf2 = avdf[['uqID','GT','Dosegrp']].copy()
# avdf3 = avdf.filter(regex='BM_')
# avdf4 = pd.DataFrame(avdf2,columns = ['GT'])
# Massdf = pd.concat([avdf3,avdf4],axis=1)


# GTgrp = []
# for value in Massdf["GT"]:
#     if value == 1:
#         GTgrp.append("WT")
#     elif value == 2:
#         GTgrp.append('cKO')
#     else:
#         GTgrp.append("Invalid")

# # Massdf["GenGrp"] = GTgrp
# Massdf["GTgrp"] = pd.Series(GTgrp)
# Massdf = Massdf[list(Massdf.columns[~Massdf.columns.duplicated()])]
# Massdf = Massdf.fillna(0)


# display(Massdf)

# BMcols =Massdf.filter(regex='BM_')
# display(BMcols)
# sns.relplot(data=Massdf, x=BMcols, y='Experiment', hue="GT")
```

In [ ]:

```
# sns.relplot(data=tdf, x="SpO2%", y="Mass (g)", hue="Label",
# #             col="Day", col_wrap=2
#            )
```

In [ ]:

```
# cutdf1 = tdf.copy()
# # display(cutdf1)

# cutdf = cutdf1[["Day","Label","Mass (g)", "SpO2%","HR (bpm)", "Temp (C)"]]
# display(cutdf)
```

In [ ]:

```
# ndf=cutdf.copy()
# ndf["Temp_cat"] = cutdf.insert(6, "Temp_cat", "")
# # ndf["Hypothermic"] = ndf.insert(6, "Hypothermic", "")
# # ndf["Low grade"] = ndf.insert(8, "Low grade", "")
# # ndf["Febrile"] = ndf.insert(9, "Febrile", "")
# display(ndf)
```

In [ ]:

```
# x = ndf.loc["Temp_cat"]
# def Temp(x):
#     if x <= 36.4:
#         return "Hypothermic"
#     elif x >= 36.5:
#         return 'Basal'
#     elif x <= 37.6:
#         return 'Low grade'
#     elif x >=39.5:
```

```
#         return 'Febrile'
#     else:
#         return x

# tempdf = ndf['Temp_cat'].apply(Temp)
# # ndf1 = pd.concat([ndf,tempdf], axis=1)
# display(tempdf.head())
# # display(ndf1.head())
```

In [ ]:

In [ ]:

In [ ]:

In [44]:
```
# if this fails, use "pip install dirsync", and adjust the source and target paths
from dirsync import sync
source_path = r'C:\Users\klafond\OneDrive - mcw.edu\Desktop\.ipynb_checkpoints'
target_path = r'G:\Gantner Lab\software\IPYNB_backup_BNG'
sync(source_path, target_path, 'sync') #for syncing one way
```

```
dirsync finished in 0.07 seconds.
3 directories parsed, 0 files copied
```

Out[44]:
```
set()
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: