datacell_request_time_v2

June 28, 2020

1 Navigating this notebook

A markdown cells in **bold text and default size** indicate that you should execute the cell below it, ex.:

> **Rerun this cell with a new date range/s to RESET DATA:**

Many other cells should be run, but the most important and ones most likely to be rerun will be denoted with a bold title cell as shown above.

The notebook will be broken up into **definitions cells** and **execution cells**. Definiton cells can be run once at the start of each new kernel session, execution cells must be reloaded every time you want to reset a dataframe or a figure. Execution cells will be indicated by the above notation.

2 Initial Setup

Eugene: you can experiemnt with turning matplotlib inline on and off, it doesn't seem to have an effect as I think Seaborn is automatically running show() on each of the plots. But truning it on won't hurt.

[58]:

```
# Turn on inline display of plots in the notebook. (We should work on a more
# portable solution such as saving th plots to files down the road.)
%matplotlib inline
```

2.1 Create a connection function

[2]:

```
import pyodbc

# Connect via Windows authentication:
# conn_specs = ('DRIVER={ODBC Driver 17 for SQL Server};'
#                                      + 'SERVER=scotty-test;'
#                                      + 'DATABASE=Interject_Reporting;'
#                                     + 'trusted_connection=yes;')
# conn_str = 'mssql+pyodbc:///?odbc_connect={}'.format(conn_specs)
# engine = sqla.create_engine(conn_str)

# Define a function that we can use to connect to a db on the fly:
```

```
def pyodbc_connect_scotty_test(db_name):
    try:
        connection = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL Server};'
                                        + 'SERVER=scotty-test;'
                                        + 'DATABASE=' + db_name + ';'
                                       + 'trusted_connection=yes;') # Windows␣
↪auth
        return connection
    except:
        return 1 # could not connect
```

[3]:

```
import pandas as pd
```

2.2 Create a function to load requests data for specific years from SQL DB

[4]:

```
import matplotlib.pyplot as plt
import numpy as np
```

Filter years to select data:

Eugene: Robby helped me think of a way to possibly improve performance accross the entire notebook by first importing the data from the SQL DB into ”pure python” (instead of directly into a pandas dataframe) using pyodbc and putting the data first in a numpy matrix (or possibly JSON obj), then using the numpy matrix to create pandas dataframes later as they are needed. This would help because Numpy is written in C and it highly optimized for large amounts of data.

[13]:

```
def read_sql_requests_by_year(start_year: int, end_year: int, schema_name: str,␣
↪table_name_1: str, table_name_2: str) -> pd.DataFrame:

    sql = ('SELECT'
                ' [StartTimeStamp]'
                ' ,DATEDIFF(second, [StartTimeStamp], [EndTimeStamp]) AS␣
↪Duration'
            ' FROM [' + schema_name + '].[' + table_name_1 + ']'
            ' UNION ALL'
            ' SELECT'
            ' [StartTimeStamp]'
            ' ,DATEDIFF(second, [StartTimeStamp], [EndTimeStamp]) AS Duration'
            ' FROM [' + schema_name + '].[' + table_name_2 + ']'
            ' WHERE (SUBSTRING(CONVERT(NVARCHAR(10), [StartTimeStamp], 102), 1,␣
↪4) BETWEEN ' + str(start_year) + ' AND ' + str(end_year) + ')'
            ' AND ([StartTimeStamp] < [EndTimeStamp])'  # Eliminate the␣
↪possibility of negative durations
          )

    requests = pd.read_sql(sql, reporting_conn, index_col='StartTimeStamp')
    #requests.t
    return requests
```

Set the following variables to the location (database, schema, tables) of your DataCell request data:

[14]:

```
reporting_conn = pyodbc_connect_scotty_test(
    db_name = 'Interject_Reporting'
)

schema = 'test'
# Set this to the name of the table containing recent DataCell request data:
table_recent = 'Request'
# Set this to the name of the table containing historical DataCell request data:
table_hist = 'Request_History'
```

2.3 Create functions to create the 2 basis DataFrames

The following two functions work together to bring the data from the SQL database connection to the dataframes. The first brings only the raw request data (Duration and a StartTimeStamp index) into a pandas DataFrame, and the second brings in this plus some extra DF columns with integers representing Year, Month, Day, Week and a string containing Weekday name.

[15]:

```
def get_df_requests_from_db(start_year: int, end_year: int) -> pd.DataFrame:
    ''' Reload the raw request data (StartTimeStamp (DF index) and Duration)␣
 ↪from SQL DB into a
        DataFrame, for a given year range. '''

    requests = read_sql_requests_by_year(start_year, end_year, schema,␣
 ↪table_recent, table_hist)
    requests.dropna(inplace=True)  # Make sure this is what we want

    # Decrease memory usage for requests:
    requests['Duration'] = requests['Duration'].astype('float32')

    return requests
```

[16]:

```
def get_df_requests_with_date_cols_from_db(start_year: int, end_year: int) ->␣
 ↪pd.DataFrame:
    ''' Reload the request data (StartTimeStamp (DF index) and Duration) from␣
 ↪SQL DB into a
        DataFrame, and add useful date columns Year, Month, Week, Weekday.␣
 ↪Within a specified
        year range. '''

    # docstring possible addition (ignore):
```

```
#     Gives the option of passing
#     in an existing instance of requests DF (must be from a call to␣
 ↪get_df_requests_from_db())
#     or, if dataframe=None, this function will instantiate a new basis␣
 ↪requests DF.

    # BUGGGY:
#     if (dataframe is None):
#         requests_base = get_df_requests_from_db(start_year, end_year)
#         # Decrease memory usage for requests_base:
#         requests_base['Duration'] = requests_base['Duration'].
 ↪astype('float32')
#     else:
#         requests_base = dataframe

    requests_base = get_df_requests_from_db(start_year, end_year)

    # Create DF of requests with useful dates columns:
    requests_dates = requests_base
    requests_dates['Year'] = requests_base.index.year
    requests_dates['Month'] = requests_base.index.month
    requests_dates['Week'] = requests_base.index.week
    requests_dates['Weekday'] = requests_base.index.weekday_name
    requests_dates['Day'] = requests_base.index.day
#     requests_dates.dropna(inplace=True)

    # Decrease memory usage for requests_dates:
    requests_dates[['Year', 'Month', 'Week', 'Day']] = requests_dates[['Year',␣
 ↪'Month', 'Week', 'Day']].astype('int16')

    return requests_dates
```

2.4 Load intial data

The following section will show what the base DataFrames being loaded in from SQL will look like.

Rerun this cell with a new date range/s to reset the sample data:

[17]:
```
requests = get_df_requests_from_db(2018, 2019)
requests_dates = get_df_requests_with_date_cols_from_db(2018, 2019)
```

First look at requests DF

[17]:
```
requests.head()
```

```
[17]:                          Duration
      StartTimeStamp
      2019-08-01 17:00:00.947       1.0
      2019-08-01 17:00:21.560      18.0
      2019-08-01 17:00:49.377       0.0
      2019-08-01 17:01:07.030      15.0
      2019-08-01 17:03:34.497       0.0
```

```
[29]: requests.describe()
```

```
[29]:            Duration
      count  1.608252e+06
      mean   1.891486e+00
      std    1.314368e+01
      min    0.000000e+00
      25%    0.000000e+00
      50%    1.000000e+00
      75%    2.000000e+00
      max    1.343400e+04
```

```
[34]: requests.isna().sum()
```

```
[34]: Duration    2
      dtype: int64
```

Verify types are small, as intended:

```
[35]: print(type(requests.index))
      print(type(requests['Duration'][0]))
```

```
<class 'pandas.core.indexes.datetimes.DatetimeIndex'>
<class 'numpy.float32'>
```

First look at requests_dates DF

```
[36]: requests_dates.head()
```

```
[36]:                          Duration  Year  Month  Week   Weekday  Day
      StartTimeStamp
      2019-08-01 17:00:00.947       1.0  2019      8    31  Thursday    1
      2019-08-01 17:00:21.560      18.0  2019      8    31  Thursday    1
      2019-08-01 17:00:49.377       0.0  2019      8    31  Thursday    1
      2019-08-01 17:01:07.030      15.0  2019      8    31  Thursday    1
      2019-08-01 17:03:34.497       0.0  2019      8    31  Thursday    1
```

```
[66]: requests_dates.describe()
```

```
[66]:             Duration          Year         Month          Week           Day
      count   1.608257e+06  1.608257e+06  1.608257e+06  1.608257e+06  1.608257e+06
      mean    1.891479e+00  2.018331e+03  5.626664e+00  2.217515e+01  1.134353e+01
      std     1.314366e+01  4.706673e-01  3.317209e+00  1.443448e+01  7.608449e+00
      min    -1.000000e+00  2.018000e+03  1.000000e+00  1.000000e+00  1.000000e+00
      25%     0.000000e+00  2.018000e+03  3.000000e+00  1.000000e+01  5.000000e+00
      50%     1.000000e+00  2.018000e+03  5.000000e+00  2.000000e+01  9.000000e+00
      75%     2.000000e+00  2.019000e+03  8.000000e+00  3.400000e+01  1.600000e+01
      max     1.343400e+04  2.019000e+03  1.200000e+01  5.200000e+01  3.100000e+01
```

```
[37]: requests_dates.isna().sum()
```

```
[37]: Duration    2
      Year        0
      Month       0
      Week        0
      Weekday     0
      Day         0
      dtype: int64
```

Verify types are small, as intended:

```
[14]: print(type(requests_dates.index))
      print(type(requests_dates['Duration'][0]))
      print(type(requests_dates['Year'][0]))
      print(type(requests_dates['Month'][0]))
      print(type(requests_dates['Week'][0]))
      print(type(requests_dates['Weekday'][0]))
      print(type(requests_dates['Day'][0]))
```

```
<class 'pandas.core.indexes.datetimes.DatetimeIndex'>
<class 'numpy.float32'>
<class 'numpy.int16'>
<class 'numpy.int16'>
<class 'numpy.int16'>
<class 'str'>
<class 'numpy.int16'>
```

3 Create some useful plotting helper functions

3.1 Formatting functions:

Month name-number mapping for subplot (Axes) names

```
[67]: def place_month_names_on_axes(is_single_axes: bool, loc_str: str, font_size:␣
       ↪int, dataframe: pd.DataFrame =None, axes_arr: list =None, ax: plt.Axes␣
       ↪=None, df_month_col_name: str =None) -> None:
```

```
    ''' Based on whether is_single_axes is T/F, this function takes either:

            if is_single_axes True:
            A matplotlib Axes object (ax), the DataFrame being plotted (dataframe), and the name
            of the DF col containing an integer representing month of the year (df_month_col_name).

            OR

            if is_single_axes False:
            A matplotlib Axes ARRAY (axes_arr).

        The first case will place the month name on the single Axes object it is passed,
        based on the contents of dataframe[df_month_col_name].

        The second case will iterate through the array of Axes, and place the month name on each
        Axes based on the default name that matplotlib gives each axes when it is created in a
        Seaborn FacetGrid.

        The other, required, args are:
        is_single_axes: True if passing a single Axes object,
        loc_str: location ('left', 'right' or 'center') on the Axes object to place the month name),
        font_size: font size of the month name title being placed on the Axes/axis. '''

    # Create a dict of name-number pairs for month labeling:
    month_num_name = {
        1: 'January',
        2: 'February',
        3: 'March',
        4: 'April',
        5: 'May',
        6: 'June',
        7: 'July',
        8: 'August',
        9: 'September',
        10: 'October',
        11: 'November',
        12: 'December'
    }
    if (is_single_axes):
        month_number = dataframe[df_month_col_name].astype(int)[0]
        ax.set_title(month_num_name[month_number], loc=loc_str,␣
↪fontsize=font_size, pad=10)
    else:
        for month in range(len(axes_arr)):
            # Extract the two characters that will contain the number:
            this_month_str = (axes[month].get_title())[-2] + (axes[month].
↪get_title())[-1]
            # If the letter at position -4 from get_title() is a space, then it␣
↪must have been a single-digit number in the name,
            # so extract it:
            if this_month_str[0] == ' ':
                this_month_num = int(this_month_str[1])
            else:
                this_month_num = int(this_month_str)
            # Change the title to the correct month name from the dict:
            axes[month].set_title(month_num_name[this_month_num], loc=loc_str,␣
↪fontsize=font_size) # KNOWN BUG: sometimes this call will leave the old␣
↪title in place.
```

Year title formatter for subplots (Axes) titles

[68]:
```
def set_year_name_axes_titles(axes_arr: plt.Axes, dataframe: pd.DataFrame,␣
↪loc_str: str, df_year_name_col: str) -> None:
    ''' This function takes an array of matplotlib Axes objects and places a␣
↪year title on each of them.

        dataframe: dataframe being plotted.
        df_year_name_col: the name of the column containing the year integer.␣
↪DF col must contain the full,
            4-digit year idetifier.'''

    for year in range(dataframe[df_year_name_col].nunique()):
        current_title = axes[year].get_title()
#         axes[year].set_title(' ')
        new_title = current_title[7:11]
        axes[year].set_title(new_title, loc=loc, fontsize=16)
```

3.2 DataFrame generator functions:

The following functions generate new DFs either based on filtering another DF being passed in, or create a new DF using get_df_requests_with_date_cols_from_db() and then filtering the resulting dataframe further.

[69]:

```
def get_df_no_weekends(dataframe: pd.DataFrame, df_weekday_name_col: str) -> pd.
↪DataFrame:
    ''' Takes a dataframe and returns a copy of that dataframe without any␣
↪records observed on a weekend.

        dataframe: the basis dataframe to remove weekends from in returned copy.
        df_weekday_name_col: the name of the column containing the Weekday name.
↪ DF col must contain strings
            with no extra spaces, first letter capitalized. '''

    weekday_filter = ( (dataframe[df_weekday_name_col] != 'Saturday') &␣
↪(dataframe[df_weekday_name_col] != 'Sunday') )
    dataframe_no_weekends = dataframe.where(weekday_filter)
    dataframe_no_weekends = dataframe_no_weekends.dropna()
    return dataframe_no_weekends
```

[387]:

```
def get_df_requests_1_month(year: int, month: int, df_month_name_col: str) ->␣
↪pd.DataFrame:
    ''' Utilizes get_df_requests_with_date_cols_from_db() to pull in data from␣
↪DB for a specific year,
        then further filters this DF by month.

        year: year to filter by.
        month: month to filter by.
        df_month_name_col: the name of the column containing the month␣
↪identifier, where the month
            identifier is an integer. '''

    requests_dates_year = get_df_requests_with_date_cols_from_db(year, year)

    single_month_filter = ( requests_dates_year[df_month_name_col] == month )
    requests_dates_month = requests_dates_year.where(single_month_filter)
    requests_dates_month.dropna(inplace=True)

    return requests_dates_month
```

[388]:

```
def get_df_requests_month_range(year: int, start_month: int, end_month: int,␣
↪df_month_name_col: str) -> pd.DataFrame:
    ''' Utilizes get_df_requests_with_date_cols_from_db() to pull in data from␣
↪DB for a specific year,
        then further filters this DF by the specified month range.

        year: year to filter by.
        start_month: first month in range to filter by, inclusive.
        end_month: last month to filter by, inclusive.
        df_month_name_col: the name of the column containing the month identifier, where the month
            identifier is an integer. '''

    requests_dates_year = get_df_requests_with_date_cols_from_db(year, year)

    month_filter = ( (requests_dates_year[df_month_name_col] >= start_month) & (requests_dates_year[df_month_name_col] <= end_month) ) # Boolean series
    requests_3_months = requests_dates_year.where(month_filter)
    requests_3_months.dropna(inplace=True)

    return requests_3_months
```

4 Analyses

Within this section, multiple different analyses of the DataCell request data will be constructed. There are subsections which are numbered based on the data story being told.

Section numbering: Format of titles: **#.#.# - Title**

First section (#.x.x) - The first number in section titles (#.x.x) defines the overarching type of analysis being done throughout the whole section, e.g. analysis over 1 year of data, and will be indicated in a title first by markdown heading 2 (h2). For example, subsection 1. contains multiple different analyses of request data over 1 year. Sometimes, a base DataFrame that will be used in all subsections will be defined by a function prep_base_data_#() in first subsections.

Second subsection (x.#.x) - The second subsection (x.#.x) in the title indicates the DataFrame-level subsection. This means that each distinct subsection at this level (e.g. 1.1, 1.2 or 3.4, etc.) has it's own DataFrame defined for it, and the analyses/visualizations within this subsection are all being done on that DataFrame. The DataFrames for these subsections will be created by a function named some_data_prep_description_#_#() for each subsection, which will be defined and called right after the title of the subsection. The titles for second subsections will be in markdown header 4 (h4).

Third subsection (x.x.#) - The third and final subsection defines an individual figure/plot. All third subsections will belong to a second subsection that defines the DataFrame that the figure uses. Each figure/plot will have a ”Figure definitions” section right before the figure source code where you can modify variables that effect the look or execution of the figure, and they will be labeled figure_defintion_variable_###. The titles for third subsections will be in markdown header 5 (h5).

”prep_data_#()” and similar data prep functions used in these sections: Most of the data prep functions used herein (#.x.x) are just wrapper functions that call get_df_requests_with_date_cols_from_db() for a specified year period, or one of the month-filtering get_df functions defined above, and then give the user the choice to exclude weekends or not. They are used as an information-hiding mechanism, and to put everything in once place.

Some of these data prep functions, however, also do aggregation and other pandas DataFrame manipulation and then return a new modified DF. These functions will be more interesting to read.

”clear_df_from_mem_#_#_#” variable note: This variable is in the ”Figure definitions” section for each individual figure. It is there to save memory by deleting the DataFrame that is being plotted after the figure is rendered. It should be set to **True** when you just want to render the figure and keep it up for looking at. However, if you want to edit the figure and reload it as you make changes to the figure source code or the figure definitions variables, you should set clear_df_from_mem_#_#_# to **False** so that your DataFrame will be kept in memory and you don’t have to constantly go back to the data prep function and reload the DataFrame every time you make a change to the figure.

Other notes for Eugene: I still need to add more user-control variables to the ”Figure definitions” sections on all of these plots, but I’m sure you can figure out how to change the matplotlib code.

There are still a few bugs with some of the plots that will have comments.

4.1 1 - Data over 1 year, by month

1.1 - Mean estimation of all requests, 1 year

[111]:
```
# Known bugs:
    # -With fewer Axes, the figure looks bad (figure title overlaps first Axes,␣
 ↪etc.). Need to make it dynamically styled. See fig.subplots_adjust() call.
```

[112]:
```
# Make labels readable. Exclude weekends. Exclude holidays? Make if very␣
 ↪obvious that we are excluding thesedays, make this reversible.
```

DataFrame prep:

[72]:
```
def prep_data_1_1(year: int, exclude_weekends: bool) -> pd.DataFrame:

    df_1_1 = get_df_requests_with_date_cols_from_db(2019, 2019)

    if (exclude_weekends):
        df_1_1 = drop_weekends(df_1_1)

    return df_1_1
```

Load or reload data:

[73]:
```
requests_1yr = prep_data_1_1(
    year=2019,
    exclude_weekends=False
)
```

[74]:
```
requests_1yr.head()
```

[74]:
```
                         Duration  Year  Month  Week   Weekday  Day
StartTimeStamp
2019-08-01 17:00:00.947       1.0  2019      8    31  Thursday    1
2019-08-01 17:00:21.560      18.0  2019      8    31  Thursday    1
2019-08-01 17:00:49.377       0.0  2019      8    31  Thursday    1
2019-08-01 17:01:07.030      15.0  2019      8    31  Thursday    1
2019-08-01 17:03:34.497       0.0  2019      8    31  Thursday    1
```

1.1.1 - Plot of mean estimation of all request durations over 1 year, with subplots/Axes for each month

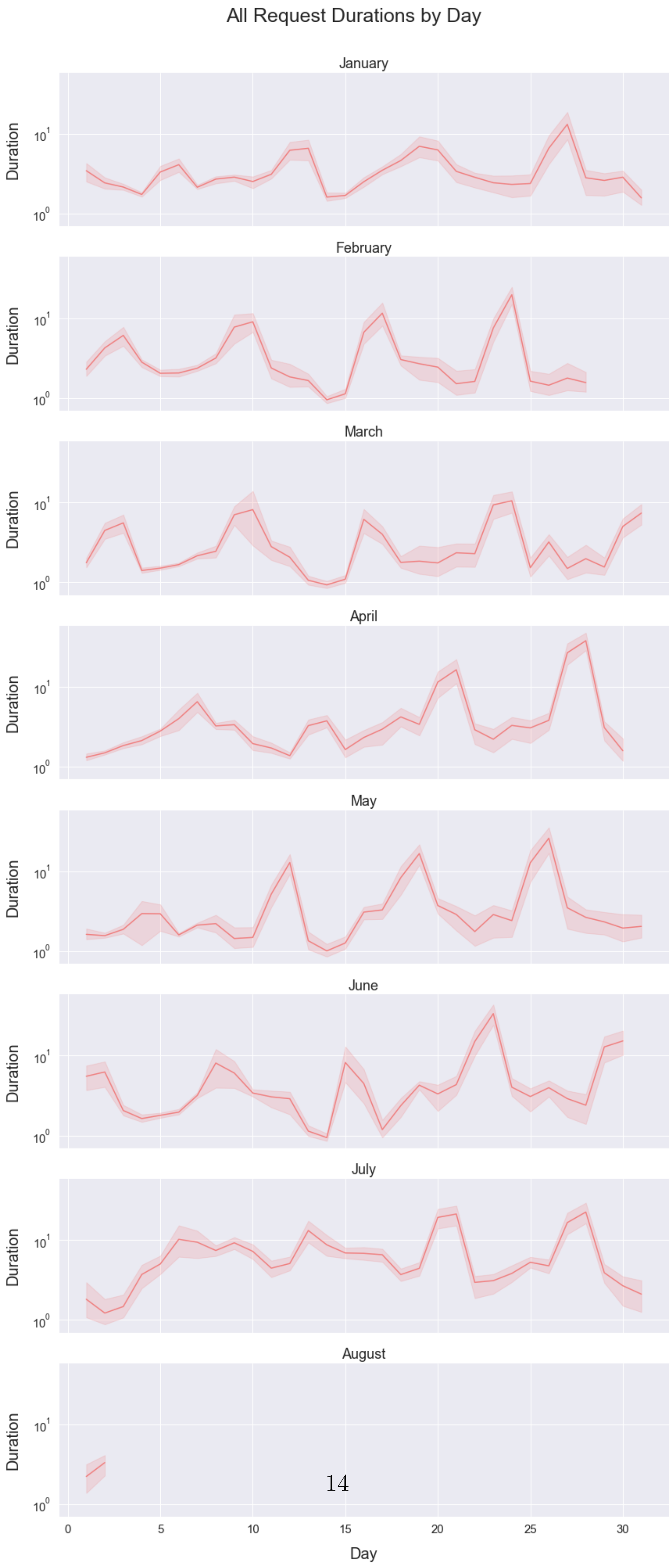
Alter figure definitions:

[75]:
```
# all_data_line_color_1_1_1 = 'steelblue'
mean_data_line_color_1_1_1 = 'lightcoral'

# Set this to True if you would like the DataFrame to be deleted from memory␣
 ↪after the figure is rendered:
clear_df_from_mem_1_1_1 = True
```

Run cell below to render the figure:

[76]:
```
import seaborn as sns

sns.set(style='darkgrid')

month_grid = sns.FacetGrid(requests_1yr, row='Month', height=3.5, aspect=4,␣
 ↪legend_out=True)

# All observations:
# month_grid.map(sns.lineplot, 'Day', 'Duration',␣
 ↪color=all_data_line_color_1_1_1)
# Mean for each day:
month_grid.map(sns.lineplot, 'Day', 'Duration', estimator='mean',␣
 ↪color=mean_data_line_color_1_1_1)

# Set figure aesthetics:

month_grid.fig.suptitle('All Request Durations by Day', fontsize=25)

month_grid.set_ylabels('Duration', fontsize=20, labelpad=15)
month_grid.set_xlabels('Day', fontsize=20, labelpad=15)

axes = month_grid.axes.flatten() # Numpy flatten 2D array of axes to array or␣
 ↪1D rows.
```

```
place_month_names_on_axes(is_single_axes=False, axes_arr=axes,␣
 ↪loc_str='center', font_size=18)

month_grid.set(yscale='symlog')  # Retain zero values
# ax.set_ylim(bottom=-0.25)  # 0 doesn't work...

# Settings that need to be applied to each axis:
for i in range(len(axes)):
    axes[i].tick_params(axis='both', which='major', labelsize=15)

# Set a figure-level legend with the lines (0 and 1) from the first ([0]) Axes␣
 ↪(can do this b/c because all Axes have the same two lines):
# month_grid.fig.legend(handles=(axes[0].lines[0], axes[0].lines[1]),␣
 ↪labels=('All Request Durations', 'Mean Request Dur. per Day'), borderpad=0.
 ↪5, labelspacing=0.8, fontsize=15, loc='upper right')

month_grid.fig.subplots_adjust(top=0.939, left=0.125, right=0.9, hspace=0.2,␣
 ↪wspace=0.2)  # These are magic numbers, do not adjust.

# month_grid.add_legend(['All Request Durations', 'Mean Request Dur. per Day'])

# month_grid.set_xticklabels('')

if (clear_df_from_mem_1_1_1):
    del requests_1yr
```


All Request Durations by Day
January
February
March
April
May
June
July
August
Duration
10^1
10^0
0
5
10
15
20
25
30
Day

1.2 - Daily request load vs. mean and max request durations by month for 1 year

[79]:
```
def prep_data_1_2(year: int, exclude_weekends: bool) -> pd.DataFrame:

    base_data_yr = get_df_requests_with_date_cols_from_db(year, year)

    if (exclude_weekends):
        base_data_yr = get_df_no_weekends(base_data_yr, 'Weekday')

    df_1_2 = base_data_yr.groupby(pd.Grouper(freq='D')).agg(
    Mean_Duration=pd.NamedAgg(column='Duration', aggfunc=np.mean),
    Max_Duration=pd.NamedAgg(column='Duration', aggfunc='max'),
    Count_Durations=pd.NamedAgg(column='Duration', aggfunc='count'),
    Day=pd.NamedAgg(column='Day', aggfunc='max'),
    Month=pd.NamedAgg(column='Month', aggfunc='max')
    )

    df_1_2.dropna(inplace=True)

    df_1_2[['Day', 'Month']] = df_1_2[['Day', 'Month']].astype('int16')

    df_1_2.sort_values(by=['Month', 'Day'], inplace=True)

    return df_1_2
```

[80]:
```
daily_stats_1yr = prep_data_1_2(
    year=2019,
    exclude_weekends=True
)
```

[81]:
```
daily_stats_1yr.head()
```

[81]:
```
                Mean_Duration  Max_Duration  Count_Durations  Day  Month
StartTimeStamp
2019-01-01           2.627397          69.0              365    1      1
2019-01-02           1.868545         187.0             3834    2      1
2019-01-03           1.751369          99.0             6210    3      1
2019-01-04           1.541320         120.0             9124    4      1
2019-01-07           1.745326         112.0            10056    7      1
```

1.2.1 - Plot of daily mean and max request duration vs. request count/load, subplots by month, 1 year of data   Alter figure definitions:

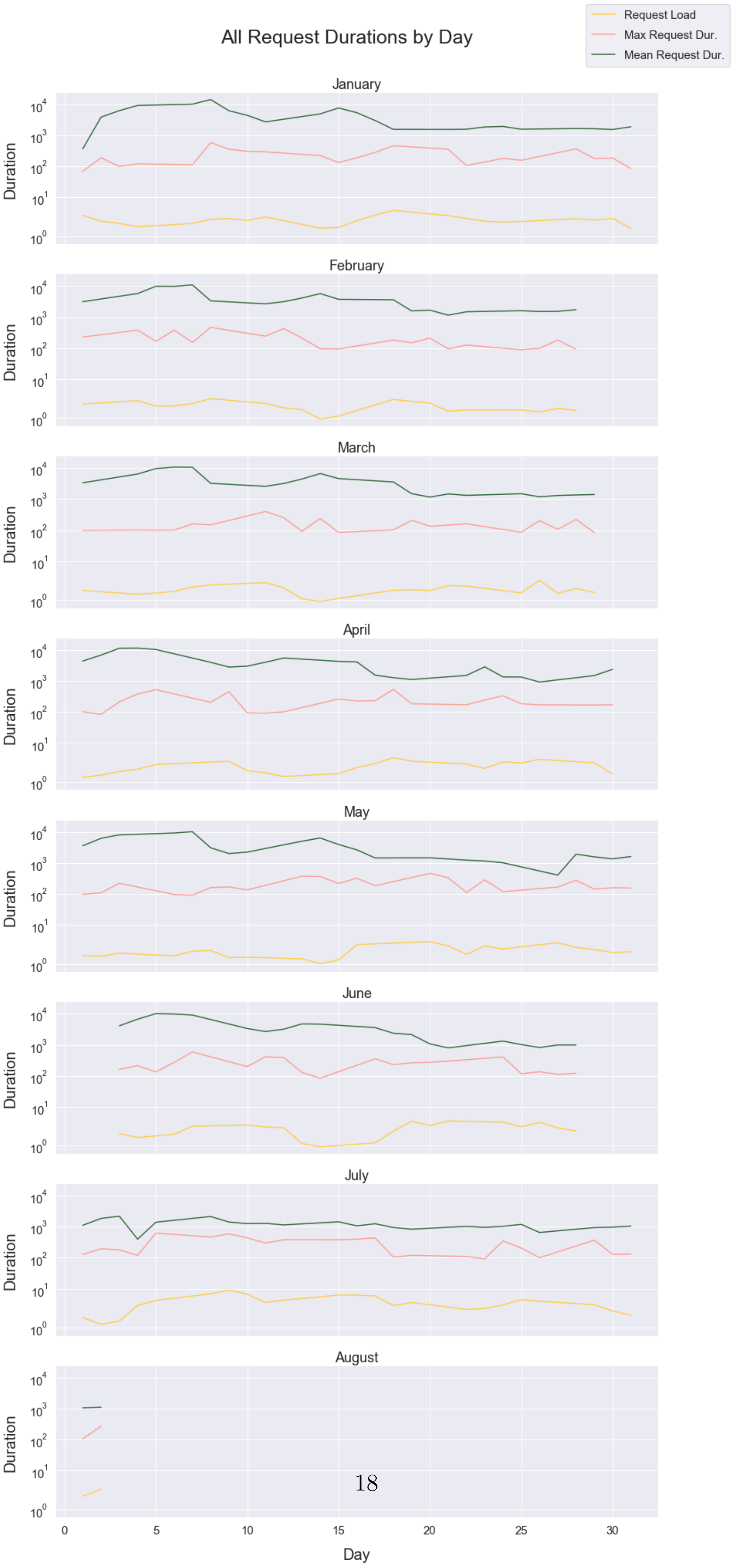
[84]:

```
mean_line_color_1_2_1 = '#ffcf4e'
max_line_color_1_2_1 = '#FFA098'
count_line_color_1_2_1 = '#386E43'

estimator_1_2_1 = 'mean'

# Set this to True if you would like the DataFrame to be deleted from memory␣
 ↪after the figure is rendered:
clear_df_from_mem_1_2_1 = True
```

[85]:

```
import seaborn as sns

sns.set(style='darkgrid')

month_grid = sns.FacetGrid(daily_stats_1yr, row='Month', height=3.5, aspect=4,␣
 ↪legend_out=True)

# All observations:
month_grid.map(sns.lineplot, 'Day', 'Mean_Duration', estimator=estimator_1_2_1,␣
 ↪color=mean_line_color_1_2_1)
month_grid.map(sns.lineplot, 'Day', 'Max_Duration', estimator=estimator_1_2_1,␣
 ↪color=max_line_color_1_2_1)
month_grid.map(sns.lineplot, 'Day', 'Count_Durations',␣
 ↪estimator=estimator_1_2_1, color=count_line_color_1_2_1)

# Set figure aesthetics:

month_grid.fig.suptitle('All Request Durations by Day', fontsize=25)

month_grid.set_ylabels('Duration', fontsize=20, labelpad=15)
month_grid.set_xlabels('Day', fontsize=20, labelpad=15)

axes = month_grid.axes.flatten() # Numpy flatten 2D array of axes to array or␣
 ↪1D rows.
place_month_names_on_axes(is_single_axes=False, axes_arr=axes,␣
 ↪loc_str='center', font_size=18)

month_grid.set(yscale='symlog')  # Retain zero values
# ax.set_ylim(bottom=-0.25)  # 0 doesn't work...

# Settings that need to be applied to each axis:
for i in range(len(axes)):
    axes[i].tick_params(axis='both', which='major', labelsize=15)

# Set a figure-level legend with the lines (0 and 1) from the first ([0]) Axes␣
 ↪(can do this b/c because all Axes have the same two lines):
```

```
month_grid.fig.legend(handles=(axes[0].lines[0], axes[0].lines[1], axes[0].lines[2]), labels=('Request Load', 'Max Request Dur.', 'Mean Request Dur.'), borderpad=0.5, labelspacing=0.8, fontsize=15, loc='upper right')

month_grid.fig.subplots_adjust(top=0.939, left=0.125, right=0.9, hspace=0.2, wspace=0.2)  # These are magic numbers, do not adjust.

# month_grid.add_legend(['All Request Durations', 'Mean Request Dur. per Day'])

# month_grid.set_xticklabels('')

if (clear_df_from_mem_1_2_1):
    del daily_stats_1yr
```


All Request Durations by Day
Request Load
Max Request Dur.
Mean Request Dur.
January
February
March
April
May
June
July
August
Duration
10^0
10^1
10^2
10^3
10^4
0
5
10
15
20
25
30
Day

As opposed to **this**:

The following was the ”old way” that I tried to solve the problem of making a separate subplot for each month of data, where I had broken it up into 12 dataframes. This is the wrong solution because it does a lot of unecessary data manipulation in pandas. I then discovered Seaborn’s FacetGrid (unsed to make the above plot), which is specifically designed to take a DataFrame and use the values in one of it’s columns to break up the data into subplots.

[ ]:

```
grouped_months = all_requests.groupby('Month')
# for name, group in grouped_months:
#     print(name)
#     print(group)

# Some months have no data, exclude these:
all_requests_jan = grouped_months.get_group(1)
all_requests_feb = grouped_months.get_group(2)
all_requests_mar = grouped_months.get_group(3)
all_requests_apr = grouped_months.get_group(4)
all_requests_may = grouped_months.get_group(5)
all_requests_jun = grouped_months.get_group(6)
all_requests_jul = grouped_months.get_group(7)
#all_requests_aug = grouped_months.get_group(8)
# all_requests_sep = grouped_months.get_group(9)
all_requests_oct = grouped_months.get_group(10)
# all_requests_nov = grouped_months.get_group(11)
# all_requests_dec = grouped_months.get_group(12)
```

[ ]:

```
#--- Make this more concise?
fig_months, axes = plt.subplots(12, 1, figsize=(25,30), sharex='all')
sns.lineplot(x='Day', y='Duration', data=all_requests_jan, ax=axes[0]).
 ↪set(title="January")
sns.lineplot(x='Day', y='Duration', data=all_requests_feb, ax=axes[1])
sns.lineplot(x='Day', y='Duration', data=all_requests_mar, ax=axes[2])
sns.lineplot(x='Day', y='Duration', data=all_requests_apr, ax=axes[3])
sns.lineplot(x='Day', y='Duration', data=all_requests_may, ax=axes[4])
sns.lineplot(x='Day', y='Duration', data=all_requests_jun, ax=axes[5])
sns.lineplot(x='Day', y='Duration', data=all_requests_jul, ax=axes[6])
#sns.lineplot(x='Day', y='Duration', data=all_requests_aug, ax=axes[7])
#sns.lineplot(x='Day', y='Duration', data=all_requests_sep, ax=axes[8])
sns.lineplot(x='Day', y='Duration', data=all_requests_oct, ax=axes[9])
#sns.lineplot(x='Day', y='Duration', data=all_requests_nov, ax=axes[10])
#sns.lineplot(x='Day', y='Duration', data=all_requests_dec, ax=axes[11])`
```

4.2 2 - Microscopic view of a single month

2.1 - All request durations for a single month

DataFrame prep:

[389]:
```
def prep_data_2_1(month: int, year: int, exclude_weekends: bool) -> pd.
 ↪DataFrame:
    df_2_1 = get_df_requests_1_month(year, month, 'Month')

    # Exclude weekends:
    if (exclude_weekends):
        df_2_1 = get_df_no_weekends(df_2_1, 'Weekday')

#     df_2_1.reset_index(inplace=True)

    return df_2_1
```

Load or reload data:

[390]:
```
requests_single_month = prep_data_2_1(
    month=7,
    year=2019,
    exclude_weekends=True)
```

[391]:
```
requests_single_month.head()
```

[391]:
```
                         Duration    Year  Month  Week    Weekday  Day
StartTimeStamp
2019-07-03 16:21:26.300       0.0  2019.0    7.0  27.0  Wednesday  3.0
2019-07-03 16:21:32.103       0.0  2019.0    7.0  27.0  Wednesday  3.0
2019-07-03 16:21:40.863       1.0  2019.0    7.0  27.0  Wednesday  3.0
2019-07-03 16:21:43.840       1.0  2019.0    7.0  27.0  Wednesday  3.0
2019-07-03 16:21:52.540       0.0  2019.0    7.0  27.0  Wednesday  3.0
```

2.1.1 - Plot of daily mean estimation of request duration, 1 month of data

Eugene: here, I changed the line on the plot that showed all requests with the estimator=None variable from a sns.lineplot() to an sns.scatterplot(). The line version was supposed to show all requests by day but the data just looked like a soundwave, all over the place... this was because it is grouping all the request data by day, since I set the x-axis to 'Day', then drawing a line through all the observations. And since there are many observations each day, ranging from 0 seconds to 100s of seconds or more, it was drawing a line from zero up to very high on the y axis *each day* and it looked very odd. Showing the individual dots for each observation helps see why it looks like this, and now we get an idea of how many observations are occuring different times for each day. You can change the scatterplot back to a line by replacing "sns.scatterplot" with "sns.lineplot".

We could also put the scatterplot on the year graphs above, but it would probably look messy and just take longer to run.

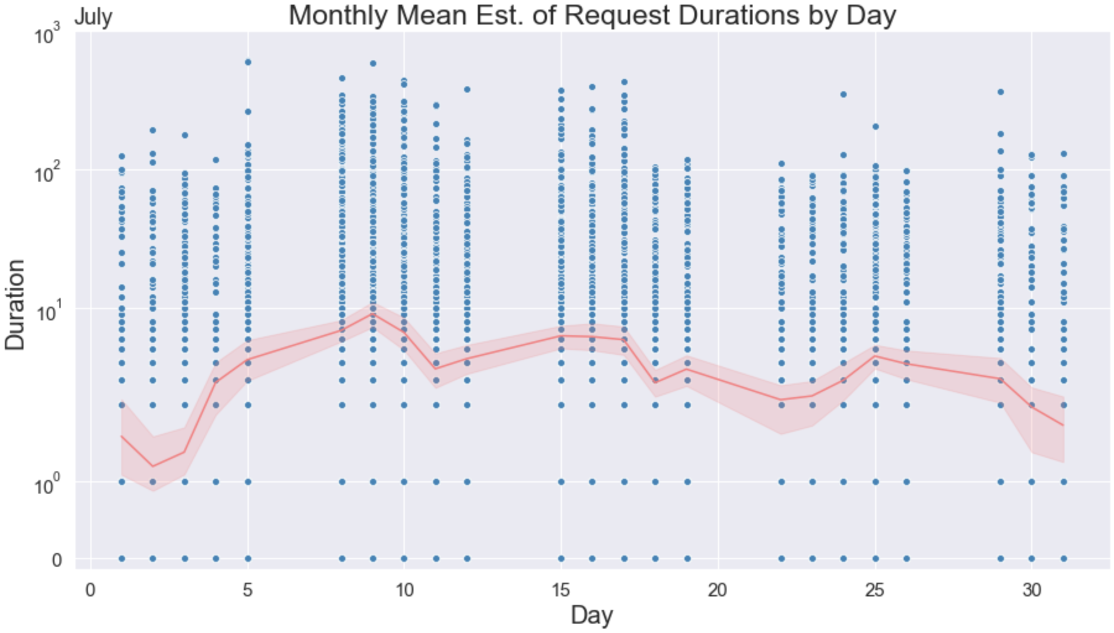
Alter figure definitions:

[375]:

```
# all_req_line_color_2_1_1 = 'steelblue'
mean_line_color_2_1_1 = 'lightcoral'

# Use this as a padding below 0 (we should not have any values under 0):
lower_y_lim_2_1_1 = -0.15

# Use this as the max request time that will be shown on the plot (as to not␣
 ↪skew the plot toward ridiculously high values):
upper_y_lim_2_1_1 = 1000

# Set this to True if you would like the DataFrame to be deleted from memory␣
 ↪after the figure is rendered:
clear_df_from_mem_2_1_1 = False
```

[376]:

```
fig, ax = plt.subplots(figsize=(15,8))
sns.set(style='darkgrid')
# sns.scatterplot(x='StartTimeStamp', y='Duration',␣
 ↪data=all_requests_single_month.reset_index())
sns.scatterplot(x='Day', y='Duration', data=requests_single_month,␣
 ↪estimator=None, color=all_req_line_color_2_1_1, ax=ax)
sns.lineplot(x='Day', y='Duration', data=requests_single_month,␣
 ↪estimator='mean', color=mean_line_color_2_1_1, ax=ax)

place_month_names_on_axes(dataframe=requests_single_month, loc_str='left',␣
 ↪font_size=18, is_single_axes=True, ax=ax, df_month_col_name='Month')

ax.set_title('Monthly Mean Est. of Request Durations by Day', fontsize=23) #␣
 ↪Change to "'January' Requests by Day"

ax.set_ylabel('Duration', fontsize=20)
ax.set_xlabel('Day', fontsize=20)

# If you would like to add back the "all_req" line, uncomment the line below:
# ax.legend(handles=(ax.lines[0], ax.lines[1]), labels=('All Request␣
 ↪Durations', 'Mean Request Dur. per Day'), borderpad=0.5, labelspacing=0.8,␣
 ↪fontsize=15, loc='upper right')


ax.tick_params(axis='both', which='major', labelsize=15)
# ax.set_xticklabels(all_requests_single_month['Day'])  # Can't use this␣
 ↪because it's taking it as a list of EACH day entry in the DF, and lineplot␣
 ↪is aggregating day values.
ax.set_yscale('symlog')
ax.set_ylim([lower_y_lim_2_1_1, upper_y_lim_2_1_1])

if (clear_df_from_mem_2_1_1):
    del requests_single_month
```

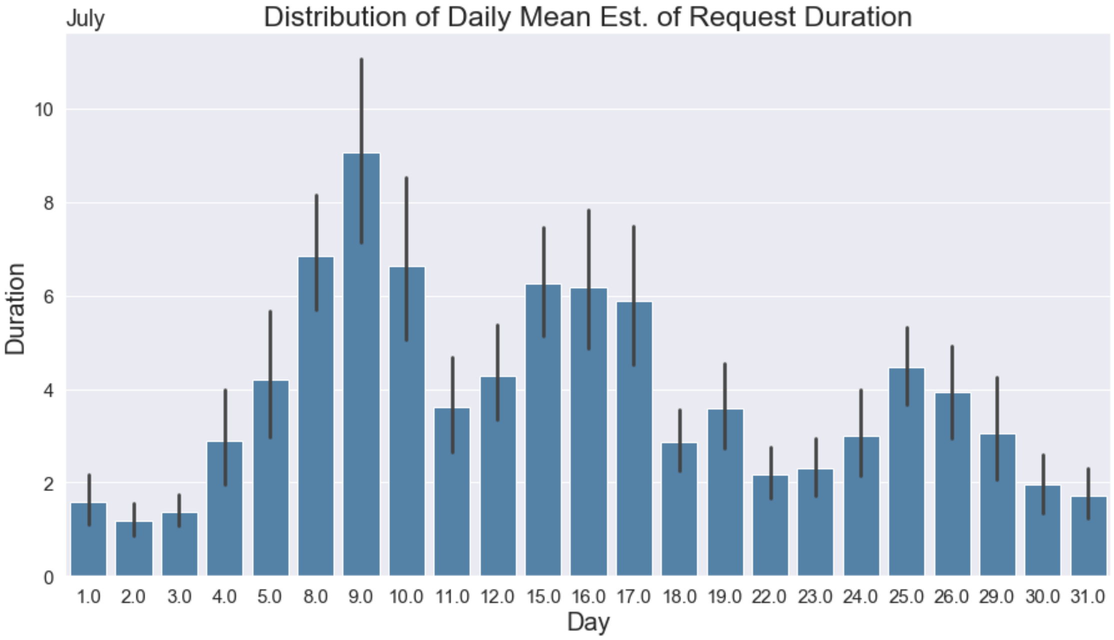


2.1.2 - Distribution of mean request time per day Alter figure definitions:

[392]:
```
all_req_line_color_2_1_2 = 'steelblue'
mean_line_color_2_1_2 = 'lightcoral'

# Use this as a padding below 0 (we should not have any values under 0):
lower_y_lim_2_1_2 = -0.15

# Use this as the max request time that will be shown on the plot (as to not␣
↪skew the plot toward ridiculously high values):
upper_y_lim_2_1_2 = 1000

# Set this to True if you would like the DataFrame to be deleted from memory␣
↪after the figure is rendered:
clear_df_from_mem_2_1_2 = False
```

[397]:
```
fig, ax = plt.subplots(figsize=(15,8))
sns.set(style='darkgrid')
sns.barplot(x='Day', y='Duration', data=requests_single_month,␣
↪color=all_req_line_color_2_1_2, ax=ax)

place_month_names_on_axes(dataframe=requests_single_month, loc_str='left',␣
↪font_size=18, is_single_axes=True, ax=ax, df_month_col_name='Month')
```

```
ax.set_title('Distribution of Daily Mean Est. of Request Duration',␣
 ↪fontsize=23) # Change to "'January' Requests by Day"

ax.set_ylabel('Duration', fontsize=20)
ax.set_xlabel('Day', fontsize=20)

ax.tick_params(axis='both', which='major', labelsize=15)

if (clear_df_from_mem_2_1_2):
    del requests_single_month
```



2.2 - Monthly stats aggregated per day

[107]:

```
def prep_data_daily_stats_2_2(month: int, year: int, exclude_weekends: bool):
    base_data_month = get_df_requests_1_month(year, month, 'Month')

    # Exclude weekends:
    if (exclude_weekends):
        base_data_month = get_df_no_weekends(base_data_month, 'Weekday')

    df_2_2 = base_data_month.groupby(pd.Grouper(freq='D')).agg(
    Min_Duration=pd.NamedAgg(column='Duration', aggfunc='min'),
    Max_Duration=pd.NamedAgg(column='Duration', aggfunc='max'),
    Mean_Duration=pd.NamedAgg(column='Duration', aggfunc=np.mean),
    Count_Durations=pd.NamedAgg(column='Duration', aggfunc='count'),
    Day=pd.NamedAgg(column='Day', aggfunc='max'),
    Month=pd.NamedAgg(column='Month', aggfunc='max')
#     Weekday=pd.NamedAgg(column='Duration', aggfunc=np.first)
    )

    df_2_2.dropna(inplace=True)

    # Convert date columns to integers so that a) they take up less space and␣
 ↪b) so that they work with "place_month_names_on_axes()"
    df_2_2[['Day', 'Month']] = df_2_2[['Day', 'Month']].astype('int16')

    return df_2_2
```

[108]:

```
daily_stats_month = prep_data_daily_stats_2_2(
    month=4,
    year=2018,
    exclude_weekends=True
)
```

[109]:

```
daily_stats_month.head()
```

[109]:

```
                Min_Duration  Max_Duration  Mean_Duration  Count_Durations  \
StartTimeStamp
2018-04-02               0.0          76.0       1.514843             3638
2018-04-03               0.0          46.0       1.425372             6653
2018-04-04               0.0         164.0       1.569870             8430
2018-04-05               0.0          46.0       1.478043            11386
2018-04-06               0.0          91.0       1.641034            11912

                Day  Month
StartTimeStamp
2018-04-02        2      4
2018-04-03        3      4
2018-04-04        4      4
2018-04-05        5      4
2018-04-06        6      4
```

2.2.1 - Plot of daily mean, max, and min of req. durations and req. load, for 1 month

Alter figure definitions:

[111]:

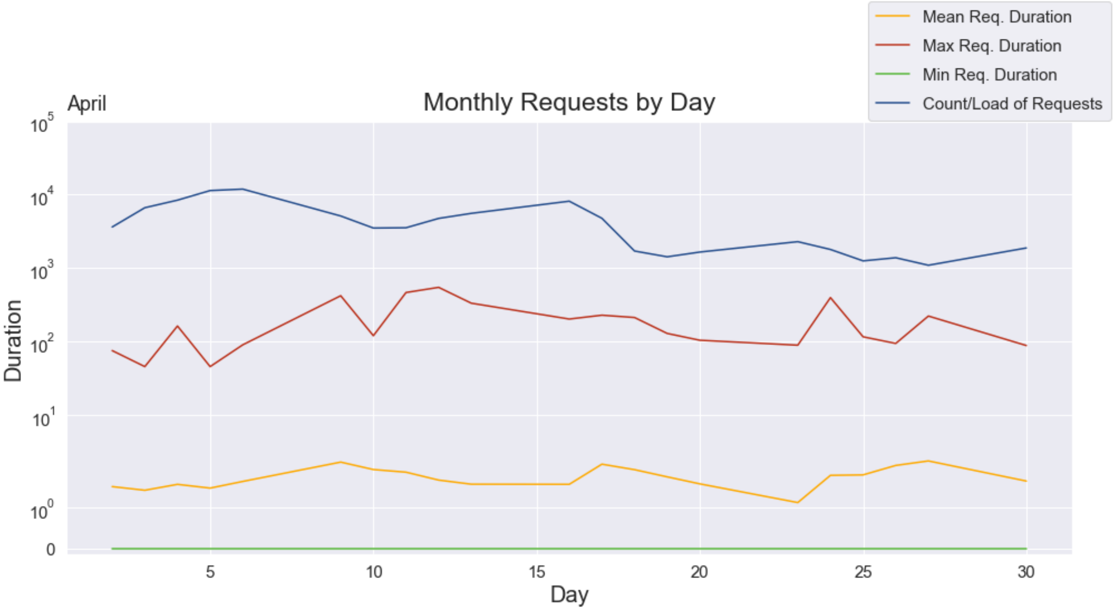
```
marker_size_2_2_1 = 60

mean_line_color_2_2_1 = '#ffaa00'
max_line_color_2_2_1 = '#bb311b'
min_line_color_2_2_1 = '#54b536'
count_line_color_2_2_1 = '#224B8B'
```

```
# Use this as a padding below 0 (we should not have any values under 0):
lower_y_lim = -0.15

# Use this as the max request time that will be shown on the plot (as to not␣
 ↪skew the plot toward ridiculously high values):
upper_y_lim = 100000

# Set this to True if you would like the DataFrame to be deleted from memory␣
 ↪after the figure is rendered:
clear_df_from_mem_2_2_1 = True
```

[112]:

```
fig, ax = plt.subplots(figsize=(15,8))
sns.set(style='darkgrid')
# sns.scatterplot(x='StartTimeStamp', y='Duration',␣
 ↪data=all_requests_single_month.reset_index())
sns.lineplot(x='Day', y='Mean_Duration', data=daily_stats_month,␣
 ↪estimator=None, color=mean_line_color_2_2_1)
sns.lineplot(x='Day', y='Max_Duration', data=daily_stats_month, estimator=None,␣
 ↪color=max_line_color_2_2_1)
sns.lineplot(x='Day', y='Min_Duration', data=daily_stats_month, estimator=None,␣
 ↪color=min_line_color_2_2_1)
sns.lineplot(x='Day', y='Count_Durations', data=daily_stats_month,␣
 ↪estimator=None, color=count_line_color_2_2_1)

ax.set_title('Monthly Requests by Day', fontsize=23) # Change to "'January'␣
 ↪Requests by Day"

place_month_names_on_axes(dataframe=daily_stats_month, loc_str='left',␣
 ↪font_size=18, is_single_axes=True, ax=ax, df_month_col_name='Month')

ax.set_ylabel('Duration', fontsize=20)
ax.set_xlabel('Day', fontsize=20)

fig.legend(handles=(ax.lines[0], ax.lines[1], ax.lines[2], ax.lines[3]),␣
 ↪labels=('Mean Req. Duration', 'Max Req. Duration', 'Min Req. Duration',␣
 ↪'Count/Load of Requests', ), borderpad=0.5, labelspacing=0.8, fontsize=15,␣
 ↪loc='upper right')

ax.tick_params(axis='both', which='major', labelsize=15)
# ax.set_xticklabels(all_requests_single_month['Day'])  # Can't use this␣
 ↪because it's taking it as a list of EACH day entry in the DF, and lineplot␣
 ↪is aggregating day values.
ax.set_yscale('symlog')
ax.set_ylim([lower_y_lim, upper_y_lim])
```

```
fig.subplots_adjust(top=0.8, left=0.125, right=0.96, hspace=0.2, wspace=0.2)  #␣
↪These are magic numbers, do not adjust.

if (clear_df_from_mem_2_2_1):
    del daily_stats_month
```



4.3 3. Analyses of request durations and load over 3 months

Consider eliminating months that we don't need in sql...

Data prep:

```
[243]: def prep_base_data_3(start_month: int, end_month: int, year: int,␣
↪exclude_weekends: bool) -> pd.DataFrame:

    # Select only month and year specified:
    base_df_3 = get_df_requests_month_range(year, start_month, end_month,␣
↪'Month')

    # Exclude weekends:
    if (exclude_weekends):
        base_df_3 = get_df_no_weekends(base_df_3, 'Weekday')

    return base_df_3
```

3.1 - Lineplots of *weekly* request data, over 3 months Function to aggregate the weekly data:

[244]:

```
def prep_weekly_data_3_1(start_month: int, end_month: int, year: int,␣
 ↪exclude_weekends: bool) -> pd.DataFrame:

    base_df_3 = prep_base_data_3(start_month, end_month, year, exclude_weekends)

    # Aggregate Duration for min, max and count over each week (using named␣
 ↪aggregation, which drops all non-aggregated, non-index cols):
    week_aggs = base_df_3.groupby(pd.Grouper(freq='W')).agg(
    Min_Duration=pd.NamedAgg(column='Duration', aggfunc='min'),
    Max_Duration=pd.NamedAgg(column='Duration', aggfunc='max'),
#     Mean_Duration=pd.NamedAgg(column='Duration', aggfunc=np.mean),
    Count_Durations=pd.NamedAgg(column='Duration', aggfunc='count'),
    Week=pd.NamedAgg(column='Week', aggfunc='max'),
    First_Day=pd.NamedAgg(column='Day', aggfunc='min'),
    Last_Day=pd.NamedAgg(column='Day', aggfunc='max')
#     Weekday=pd.NamedAgg(column='Duration', aggfunc=np.first)
    )

    # Do the last aggregation that was left out above with a simple groupby()␣
 ↪and save in a second dataframe:
    week_agg_mean = base_df_3.groupby(pd.Grouper(freq='W')).mean()
    week_agg_mean = week_agg_mean.rename(columns={'Duration':'Mean_Duration'})
    week_agg_mean.drop(axis=1, columns=['Day'], inplace=True)
    week_agg_mean.dropna(inplace=True) # use this form elsewhere!

    # Now there are 2 DFs that need to merge:
    df_3_1 = pd.merge(week_aggs, week_agg_mean, on='Week')
    # Add a "Week_Label" column that can be used as a label for the plot,␣
 ↪describing the month and the day-range of the given week:
    df_3_1['Week_Label'] = df_3_1['Year'].astype(int).astype(str) + '/' +␣
 ↪df_3_1['Month'].astype(int).astype(str) + ' days: ' + df_3_1['First_Day'].
 ↪astype(int).astype(str) + '-' + df_3_1['Last_Day'].astype(int).astype(str)
    # Drop columns that we only needed temporarily, to create the Week_Label␣
 ↪above:
    df_3_1.drop(axis=1, columns=['First_Day', 'Last_Day', 'Year', 'Month'],␣
 ↪inplace=True)

    # Lastly, reset the indexes for both final DFs back to the default integer␣
 ↪index, so that the 'Week' column can be used by plotting functions without␣
 ↪complaints:
    df_3_1.reset_index(inplace=True)
#     longform_dur_stats_3_mon_by_wk.reset_index(inplace=True)

    return df_3_1
```

Load or reload data:

```
[246]: # RUN THIS CELL TO RESET DATA:
       request_stats_3_mo_wk = prep_weekly_data_3_1(
           start_month=6,
           end_month=8,
           year=2019,
           exclude_weekends=True)
```

View the DataFrame that will be plotted:

```
[349]: request_stats_3_mo_wk
```

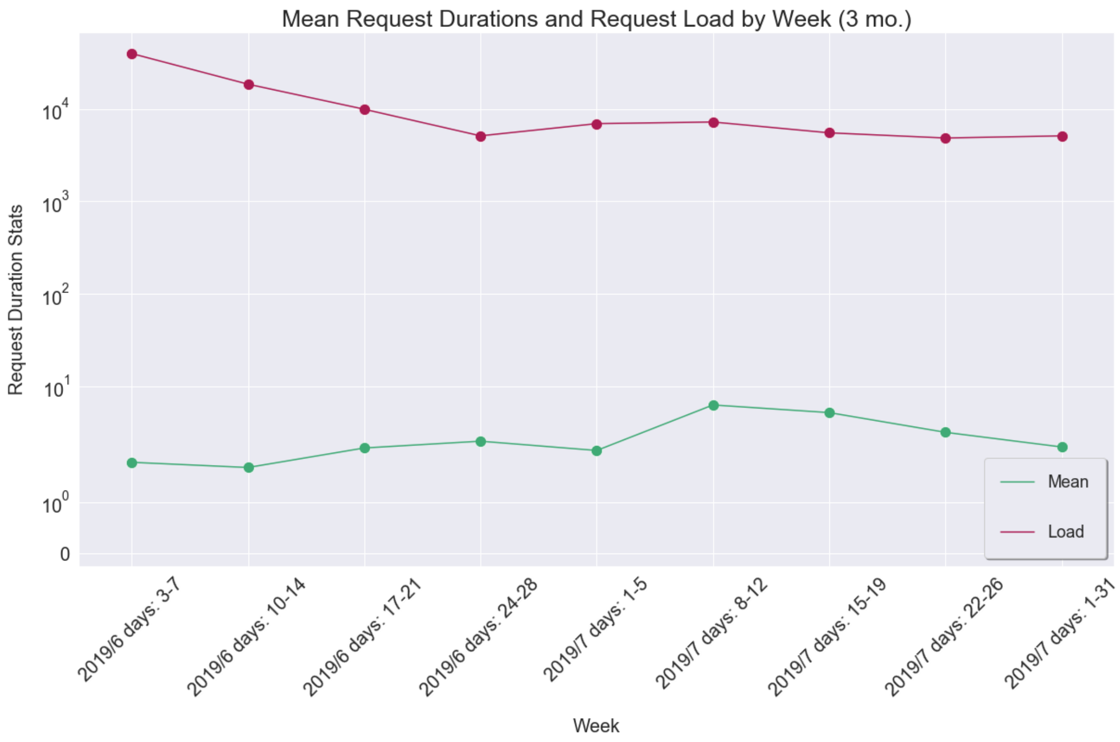
```
[349]:    index  Min_Duration  Max_Duration  Count_Durations  Week  Mean_Duration  \
       0      0           0.0         588.0            39789  23.0       1.775290
       1      1           0.0         412.0            18557  24.0       1.674301
       2      2           0.0         353.0             9937  25.0       2.144611
       3      3           0.0         407.0             5121  26.0       2.539348
       4      4           0.0         621.0             6939  27.0       2.014844
       5      5           0.0         587.0             7229  28.0       6.263660
       6      6           0.0         436.0             5520  29.0       5.178080
       7      7           0.0         349.0             4851  30.0       3.168419
       8      8           0.0         369.0             5118  31.0       2.198710

                   Week_Label
       0    2019/6 days: 3-7
       1  2019/6 days: 10-14
       2  2019/6 days: 17-21
       3  2019/6 days: 24-28
       4    2019/7 days: 1-5
       5   2019/7 days: 8-12
       6  2019/7 days: 15-19
       7  2019/7 days: 22-26
       8   2019/7 days: 1-31
```

3.1.1 - Lineplot of mean request time vs. request load/count for each week **Alter figure definitions:**

```
[249]: marker_size_3_1_1 = 150

       mean_line_color_3_1_1 = '#3FAA75'
       count_line_color_3_1_1 = '#AB1B53'

       # Set this to True if you would like the DataFrame to be deleted from memory␣
        ↪after the figure is rendered:
       clear_df_from_mem_3_1_1 = False
```

[250]:

```
sns.set_style('darkgrid')

fig, ax = plt.subplots(figsize=(20,10))

# nice green: #3FAA75
sns.lineplot(x='Week', y='Mean_Duration', data=request_stats_3_mo_wk,␣
 ↪color=mean_line_color_3_1_1, ax=ax)
sns.scatterplot(x='Week', y='Mean_Duration', data=request_stats_3_mo_wk,␣
 ↪color=mean_line_color_3_1_1, s=marker_size_3_1_1, ax=ax)
sns.lineplot(x='Week', y='Count_Durations', data=request_stats_3_mo_wk,␣
 ↪color=count_line_color_3_1_1, ax=ax)
sns.scatterplot(x='Week', y='Count_Durations', data=request_stats_3_mo_wk,␣
 ↪color=count_line_color_3_1_1, s=marker_size_3_1_1, ax=ax)

fig.subplots_adjust(top=0.9, bottom=0.1, left=0.125, right=0.9, hspace=0.2,␣
 ↪wspace=0.2) # This is magic code...

ax.set_yscale('symlog')  # Retain zero values
ax.set_ylim(bottom=-0.25)  # 0 doesn't work...

ax.set_ylabel('Request Duration Stats', fontsize=20, labelpad=15)
ax.set_xlabel('Week', fontsize=20, labelpad=20)

ax.set_title('Mean Request Durations and Request Load by Week (3 mo.)',␣
 ↪loc='center', fontsize=25)

ax.set_xticks(request_stats_3_mo_wk['Week'])
ax.set_xticklabels(request_stats_3_mo_wk['Week_Label'], rotation=45)
ax.tick_params(axis='both', which='major', labelsize=20)

ax.legend(('Mean', 'Load'), borderpad=1, labelspacing=2, fontsize=18,␣
 ↪shadow=True)

if (clear_df_from_mem_3_1_1):
    del request_stats_3_mo_wk
```



3.1.2 - Lineplot of all weekly stats, over 3 months Alter figure definitions:

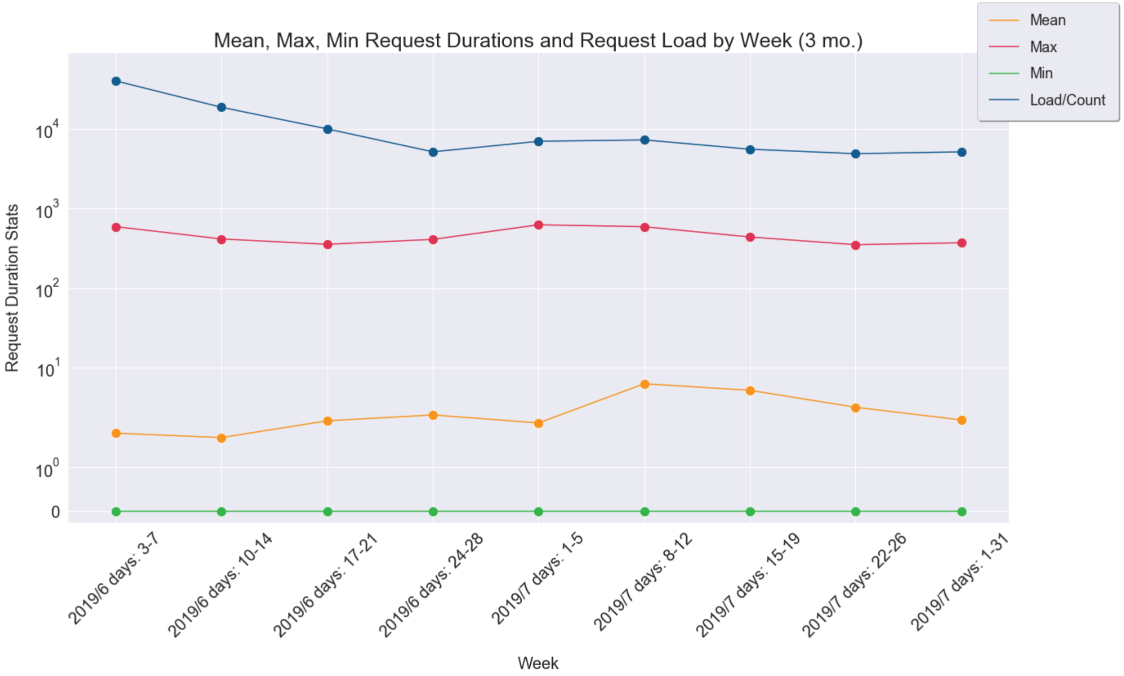
[251]:

```
marker_size_3_1_2 = 150

mean_line_color_3_1_2 = '#F7941D'
max_line_color_3_1_2 = '#DE3354'
min_line_color_3_1_2 = '#36B449'
count_line_color_3_1_2 = '#135A8E'

# Set this to True if you would like the DataFrame to be deleted from memory␣
 ↪after the figure is rendered:
clear_df_from_mem_3_1_2 = False
```

[252]:

```
sns.set_style('darkgrid')

fig, ax = plt.subplots(figsize=(20,10))

# nice green: #3FAA75
sns.lineplot(x='Week', y='Mean_Duration', data=request_stats_3_mo_wk,␣
 ↪color=mean_line_color_3_1_2, ax=ax)
sns.scatterplot(x='Week', y='Mean_Duration', data=request_stats_3_mo_wk,␣
 ↪color=mean_line_color_3_1_2, s=marker_size_3_1_2, ax=ax)
```

```
sns.lineplot(x='Week', y='Max_Duration', data=request_stats_3_mo_wk,␣
 ↪color=max_line_color_3_1_2, ax=ax)
sns.scatterplot(x='Week', y='Max_Duration', data=request_stats_3_mo_wk,␣
 ↪color=max_line_color_3_1_2, s=marker_size_3_1_2, ax=ax)

sns.lineplot(x='Week', y='Min_Duration', data=request_stats_3_mo_wk,␣
 ↪color=min_line_color_3_1_2, ax=ax)
sns.scatterplot(x='Week', y='Min_Duration', data=request_stats_3_mo_wk,␣
 ↪color=min_line_color_3_1_2, s=marker_size_3_1_2, ax=ax)

sns.lineplot(x='Week', y='Count_Durations', data=request_stats_3_mo_wk,␣
 ↪color=count_line_color_3_1_2, ax=ax)
sns.scatterplot(x='Week', y='Count_Durations', data=request_stats_3_mo_wk,␣
 ↪color=count_line_color_3_1_2, s=marker_size_3_1_2, ax=ax)

fig.subplots_adjust(top=0.9, bottom=0.1, left=0.1, right=0.9, hspace=0.2,␣
 ↪wspace=0.2) # This is magic code...

ax.set_yscale('symlog')  # Retain zero values
ax.set_ylim(bottom=-0.25)  # 0 doesn't work...

ax.set_ylabel('Request Duration Stats', fontsize=20, labelpad=15)
ax.set_xlabel('Week', fontsize=20, labelpad=20)

ax.set_title('Mean, Max, Min Request Durations and Request Load by Week (3 mo.
 ↪)', loc='center', fontsize=25)

ax.set_xticks(request_stats_3_mo_wk['Week'])
ax.set_xticklabels(request_stats_3_mo_wk['Week_Label'], rotation=45)
ax.tick_params(axis='both', which='major', labelsize=20)

fig.legend((ax.lines[0], ax.lines[1], ax.lines[2], ax.lines[3]), ('Mean',␣
 ↪'Max', 'Min', 'Load/Count'), borderpad=0.8, labelspacing=0.8, fontsize=18,␣
 ↪shadow=True)
# ax.get_legend().set_visible(False)

if (clear_df_from_mem_3_1_2):
    del request_stats_3_mo_wk
```



3.2 - Comparison of individual weekly stats, over 3 months

Data prep:

[253]:
```
def prep_longform_weekly_data_3_2(start_month: int, end_month: int, year: int,␣
↪exclude_weekends: bool) -> pd.DataFrame:

    base_data_agged_wk = prep_weekly_data_3_1(start_month, end_month, year,␣
↪exclude_weekends)

    # Create a long-form/unpivoted version of dur_stats_3_mon_by_wk that can be␣
↪used with Seaborn's swarmplot:
    df_3_2 = pd.melt(base_data_agged_wk.reset_index(), id_vars='Week',␣
↪value_vars=['Min_Duration', 'Max_Duration', 'Mean_Duration',␣
↪'Count_Durations'], var_name='Dur_Stat_Type', value_name='Dur_Stat_Value')
    df_3_2 = pd.merge(df_3_2, base_data_agged_wk[['Week_Label', 'Week']],␣
↪how='left', on='Week')
    # longform_dur_stats_3_mon_by_wk.drop(axis=1, columns=['Min_Duration',␣
↪'Max_Duration', 'Mean_Duration', 'Count_Durations', 'Year', 'Month', 'Day'],␣
↪inplace=True)

        # Convert date columns to integers so that a) they take up less space␣
↪and b) so that they work with "place_month_names_on_axes()"
    df_3_2[['Week']] = df_3_2[['Week']].astype('int16')

    return df_3_2
```

Load/reload data:

[254]:
```
# RUN THIS CELL TO RESET DATA:
lf_request_stats_3_mo_wk = prep_longform_weekly_data_3_2(
    start_month=1,
    end_month=3,
    year=2019,
    exclude_weekends=True)
```

View the DataFrame that will be plotted:

[347]:
```
lf_request_stats_3_mo_wk.head()
```

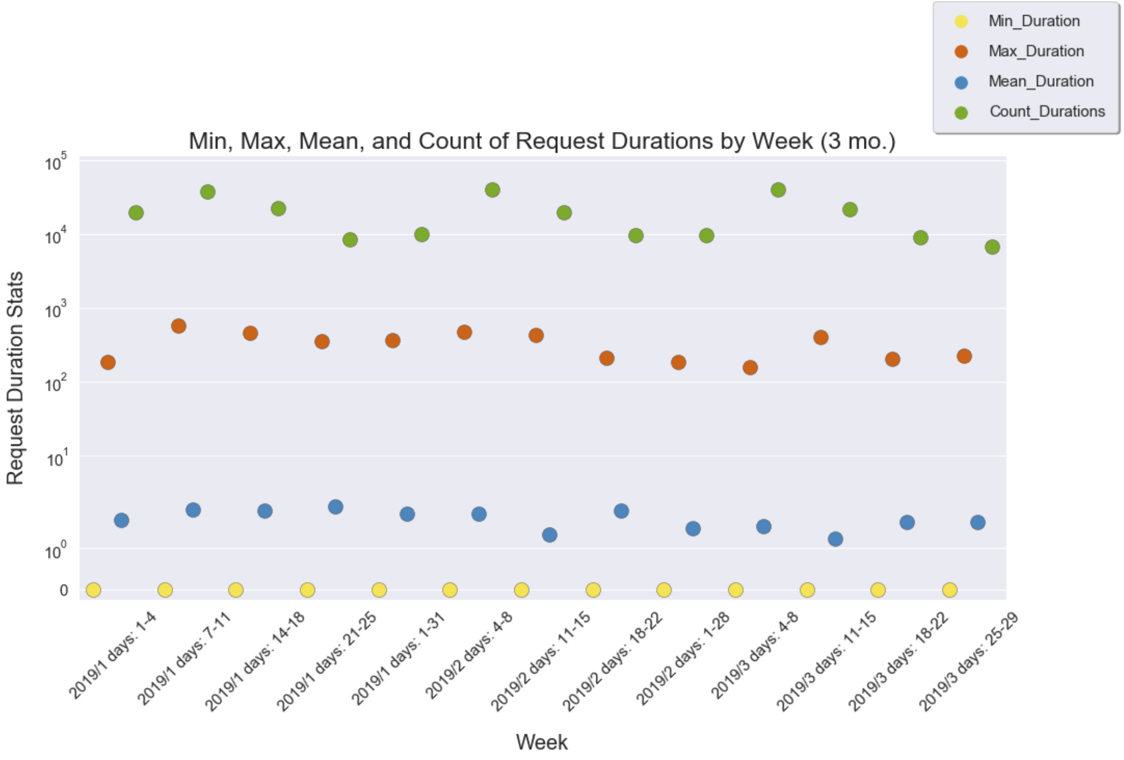
[347]:
```
   Week Dur_Stat_Type  Dur_Stat_Value          Week_Label
0     1  Min_Duration             0.0    2019/1 days: 1-4
1     2  Min_Duration             0.0   2019/1 days: 7-11
2     3  Min_Duration             0.0  2019/1 days: 14-18
3     4  Min_Duration             0.0  2019/1 days: 21-25
4     5  Min_Duration             0.0   2019/1 days: 1-31
```

3.2.1 - Swarmplot of WEEKLY mean, max and min req. durations and req. load, 3 months of data   Alter figure definitions:

[255]:
```
marker_size_3_2_1 = 14

mean_line_color_3_2_1 = '#4d85bd'
max_line_color_3_2_1 = '#cb6318'
min_line_color_3_2_1 = '#f5e356'
count_line_color_3_2_1 = '#7caa2d'

# Set this to True if you would like the DataFrame to be deleted from memory␣
 ↪after the figure is rendered:
clear_df_from_mem_3_2_1 = False
```

[348]:
```
sns.set_style('darkgrid')

fig, ax = plt.subplots(figsize=(16,9))

# Good color schemes: https://www.schemecolor.com/bond-unknown.php, https://www.
 ↪canva.com/learn/100-color-combinations/ number 23

colors = sns.color_palette((min_line_color_3_2_1, max_line_color_3_2_1,␣
 ↪mean_line_color_3_2_1, count_line_color_3_2_1))  # MIN, MAX, MEAN, COUNT

# This graph can be plotted with x='Week' because there are 3 entries for each␣
 ↪week (as there should be),
# one for each dur. stat type, so we want 3 observations for each week...
```

```
sns.swarmplot(x='Week', y='Dur_Stat_Value', hue='Dur_Stat_Type',␣
 ↪data=lf_request_stats_3_mo_wk, dodge=True, edgecolor='gray', linewidth=.4,␣
 ↪palette=colors, s=marker_size_3_2_1, ax=ax)
# sns.regplot(x='Week', y='Mean_Duration', data=dur_stats_3_mo_wk,␣
 ↪color='#4d85bd', scatter_kws={'s': 0}, ax=ax)

fig.subplots_adjust(top=0.94)

ax.set_yscale('symlog')  # Retain zero values
ax.set_ylim(bottom=-0.25)

ax.set_ylabel('Request Duration Stats', fontsize=20, labelpad=15)
ax.set_xlabel('Week', fontsize=20, labelpad=19)
ax.set_title('Min, Max, Mean, and Count of Request Durations by Week (3 mo.)',␣
 ↪loc='center', fontsize=23)

ax.tick_params(axis='both', which='major', labelsize=15)
ax.set_xticklabels(lf_request_stats_3_mo_wk['Week_Label'], rotation=45)

fig.subplots_adjust(top=0.76, bottom=0.1, left=0.125, right=0.9, hspace=0.2,␣
 ↪wspace=0.2) # This is magic code...
fig.legend(borderpad=0.8, labelspacing=1, fontsize=15, shadow=True,␣
 ↪markerscale=1.5)
ax.get_legend().set_visible(False)

if (clear_df_from_mem_3_2_1):
    del lf_request_stats_3_mo_wk
```



3.3 - DAILY stats over 3 months Fine-tuning done... TO DO: -Apply month-naming function and add Axes titles.

[320]:
```
def prep_daily_data_3_3(start_month: int, end_month: int, year: int, exclude_weekends: bool) -> pd.DataFrame:

    base_data_3_mo = prep_base_data_3(start_month, end_month, year, exclude_weekends)

    # Aggregate 'Duration' for min, max, count:
    df_3_3 = base_data_3_mo.groupby(pd.Grouper(freq='D')).agg(
    Min_Duration=pd.NamedAgg(column='Duration', aggfunc='min'),
    Max_Duration=pd.NamedAgg(column='Duration', aggfunc='max'),
    Mean_Duration=pd.NamedAgg(column='Duration', aggfunc=np.mean),
    Count_Durations=pd.NamedAgg(column='Duration', aggfunc='count'),
    Day=pd.NamedAgg(column='Day', aggfunc='max'),
    Month=pd.NamedAgg(column='Month', aggfunc='max'),
    Year=pd.NamedAgg(column='Year', aggfunc='max')
    )

    df_3_3.dropna(inplace=True)
    # Convert date columns to integers so that a) they take up less space and␣
 ↪b) so that they work with "place_month_names_on_axes()"
    df_3_3[['Day', 'Month', 'Year']] = df_3_3[['Day', 'Month', 'Year']].
 ↪astype('int16')

    # Add a "Day_Label" column that can be used as a label for the plot,␣
 ↪describing the month day of the datapoint:
    df_3_3['Day_Label'] = df_3_3['Year'].astype(str) + '/' + df_3_3['Month'].
 ↪astype(str) + '/' + df_3_3['Day'].astype(str)

    df_3_3.sort_values(by=['Month', 'Day'], inplace=True)

    df_3_3.reset_index(inplace=True)

#     day = df_3_3['Day']
#     new_index = [index for index, val in enumerate(day)]
#     new_index_days = [val for index, val in enumerate(day)]

    return df_3_3
```

Load or reload data:

[321]:

```
# RUN THIS CELL TO RESET DATA:
daily_request_stats_3_mo = prep_daily_data_3_3(
    start_month=6,
    end_month=8,
    year=2019,
    exclude_weekends=True)
```

Was writing this to debug the problem with there being gaps in the mean estimator range on the lineplot for plot 3.3.1:

[318]:

```
day = daily_request_stats_3_mo['Day']
new_index = [index for index, val in enumerate(day)]
vals = [val for index, val in enumerate(day)]
# new_index_dict = {'New_Index': new_index, 'Day': vals}  # Wrote this to␣
 ↪possibly merge back into the dataframe..
print(new_index_dict)
```

```
{'New_Index': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38,
39, 40, 41, 42, 43, 44], 'Day': [3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 17, 18, 19,
20, 21, 24, 25, 26, 27, 28, 1, 2, 3, 4, 5, 8, 9, 10, 11, 12, 15, 16, 17, 18, 19,
22, 23, 24, 25, 26, 29, 30, 31, 1, 2]}
```

View the DataFrame that will be plotted:

[293]:

```
daily_request_stats_3_mo.head()
```

```
[293]:    Min_Duration  Max_Duration  Mean_Duration  Count_Durations  Day  Month  \
       0           0.0         160.0       1.706644             4094    3      6
       1           0.0         160.0       1.706644             4094    3      6
       2           0.0         179.0       1.377971             2188    3      7
       3           0.0         179.0       1.377971             2188    3      7
       4           0.0         214.0       1.479659             6686    4      6

          Year Day_Label  New_Index
       0  2019  2019/6/3          0
       1  2019  2019/6/3         22
       2  2019  2019/7/3          0
       3  2019  2019/7/3         22
       4  2019  2019/6/4          1
```

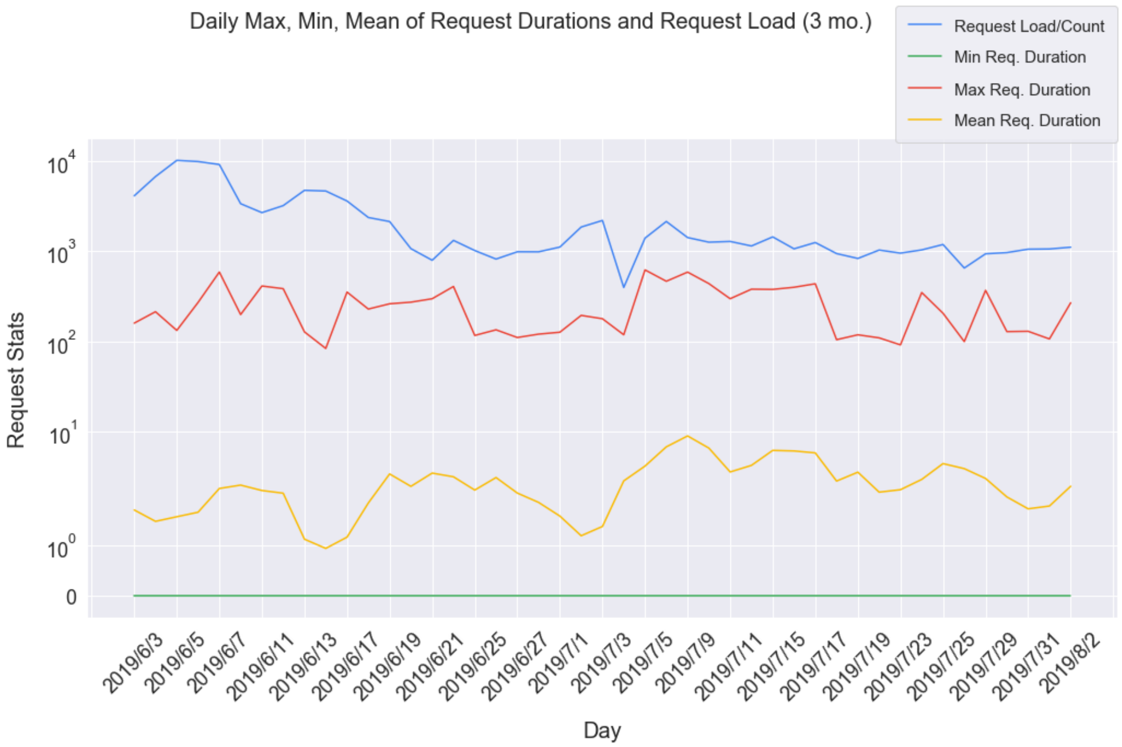
3.3.1 - Daily stats over 3-months, in single plot Eugene: This graph was producing some strange output where it would show a standard deviation from the mean but only for discrete ranges of x-values, then the std. dev graphic would not show up, then show again depending on the x-value. I figured out that because I was plotting the 'Day' column on the x-axis, it was plotting each unique DAY value, irrespective of what month/year that day was in. So instead, I reindexed the DataFrame so that there was a unique integer for each observation, then used this index as the x-value for the plot. This also fixed the problem I had with the Day_Label not matching the the day, for obvious reasons. The graph was just bucketing every observation on day "3", for example, into one x-point, which is why we had standard devs. only sometimes.

Alter figure definitions:

```
[262]: # marker_size_3_3_1 = 150

       mean_line_color_3_3_1 = '#FBBC05'
       max_line_color_3_3_1 = '#EA4335'
       min_line_color_3_3_1 = '#34A853'
       count_line_color_3_3_1 = '#4285F4'

       # Set this to True if you would like the DataFrame to be deleted from memory␣
        ↪after the figure is rendered:
       clear_df_from_mem_3_3_1 = False
```

```
[340]: from matplotlib.ticker import IndexFormatter, MultipleLocator

       fig, ax = plt.subplots(figsize=(15,9))

       sns.lineplot(data=daily_request_stats_3_mo, x=daily_request_stats_3_mo.index,␣
        ↪y='Mean_Duration', estimator='mean', color=mean_line_color_3_3_1, ax=ax)
       sns.lineplot(data=daily_request_stats_3_mo, x=daily_request_stats_3_mo.index,␣
        ↪y='Max_Duration', estimator='mean',color=max_line_color_3_3_1, ax=ax)
       sns.lineplot(data=daily_request_stats_3_mo, x=daily_request_stats_3_mo.index,␣
        ↪y='Min_Duration', estimator='mean',color=min_line_color_3_3_1, ax=ax)
```

```
sns.lineplot(data=daily_request_stats_3_mo, x=daily_request_stats_3_mo.index,␣
 ↪y='Count_Durations', estimator='mean',color=count_line_color_3_3_1, ax=ax)

ax.set_ylabel('Request Stats', fontsize=20, labelpad=15)
ax.set_xlabel('Day', fontsize=20, labelpad=15)

ax.set(yscale='symlog')  # Retain zero values
# ax.set_ylim(bottom=-0.25)  # 0 doesn't work...

# grid.set_titles(fontsize=200)

# grid.set_xticklabels('')

fig.subplots_adjust(top=0.8, left=0.125, right=0.997, hspace=0.2)  # These are␣
 ↪magic numbers, do not adjust.
fig.suptitle('Daily Max, Min, Mean of Request Durations and Request Load (3 mo.
 ↪)', fontsize=20)

fig.legend(handles=(ax.lines[3], ax.lines[2], ax.lines[1], ax.lines[0]),␣
 ↪labels=('Request Load/Count', 'Min Req. Duration', 'Max Req. Duration',␣
 ↪'Mean Req. Duration'), borderpad=0.8, labelspacing=1, fontsize=15,␣
 ↪loc='upper right')

# grid.axes[i].tick_params(axis='both', which='major', labelsize=15)

# day = daily_request_stats_3_mo['Day']
# new_index = [index for index, val in enumerate(day)]
# vals = [val for index, val in enumerate(day)]
# new_index_dict = {'New_Index': new_index, 'Day': vals}
# print(new_index)

# new_index_df = pd.DataFrame(new_index_dict)

ax.set_xticks(daily_request_stats_3_mo.index)
# ax.set_xticklabels(daily_request_stats_3_mo['Day_Label'])
ax.xaxis.set_major_locator(MultipleLocator(2))  # This sets the labels the␣
 ↪display for every other day
# ax.xaxis.set_minor_locator(MultipleLocator(2))
ax.xaxis.
 ↪set_major_formatter(IndexFormatter(daily_request_stats_3_mo['Day_Label']))

ax.tick_params(axis='x', labelsize=18, rotation=45)
ax.tick_params(axis='y', labelsize=18)

if (clear_df_from_mem_3_3_1):
    del daily_request_stats_3_mo
```



3.3.2 - Daily stats over 3 months, with plot for each month

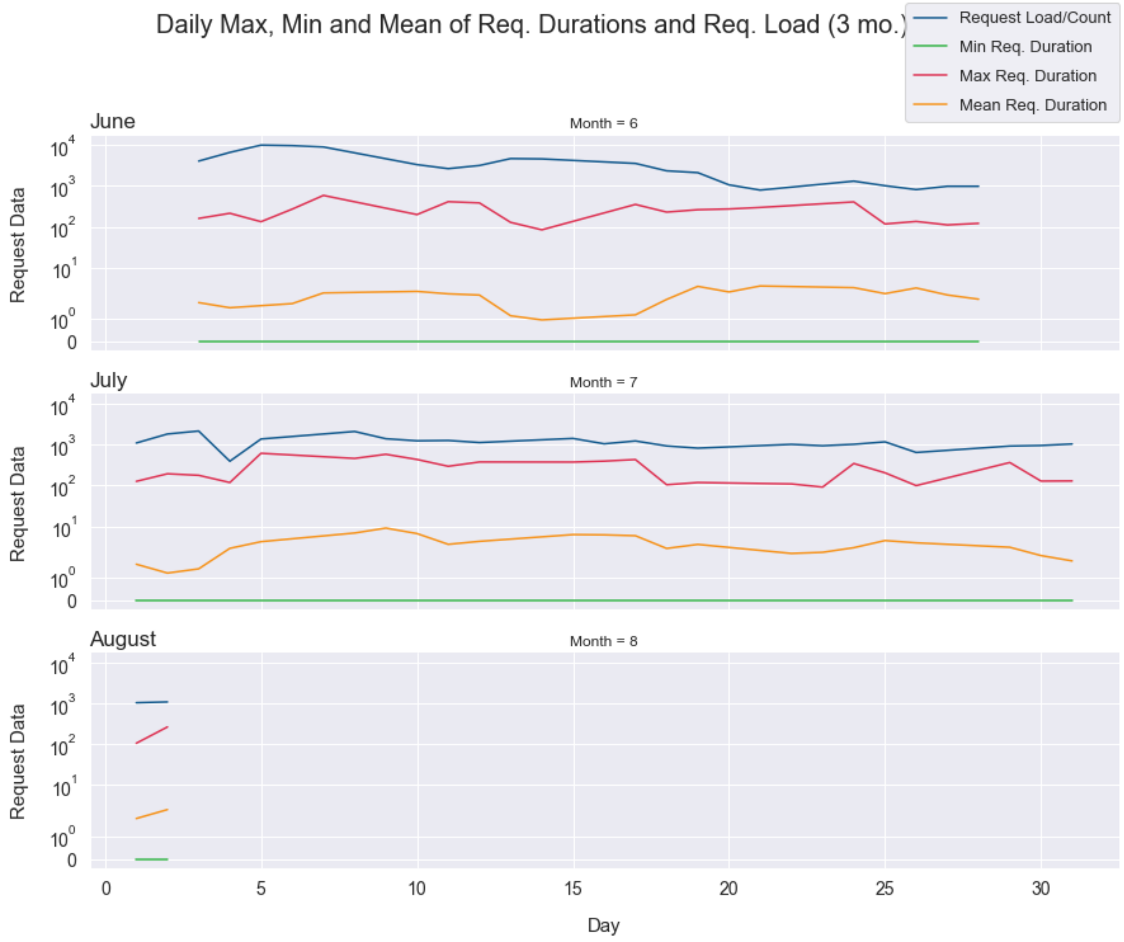
Alter figure definitions:

[152]:

```
# marker_size_3_3_2 = 150

mean_line_color_3_3_2 = '#F7941D'
max_line_color_3_3_2 = '#DE3354'
min_line_color_3_3_2 = '#36B449'
count_line_color_3_3_2 = '#135A8E'

# Set this to True if you would like the DataFrame to be deleted from memory␣
 ↪after the figure is rendered:
clear_df_from_mem_3_3_2 = False
```

[341]:

```
grid = sns.FacetGrid(row='Month', data=daily_request_stats_3_mo, height=3.4,␣
 ↪aspect=3.8, legend_out=True)

# The only reason using 'Day' isn't a problem here is that we are breaking the␣
 ↪graph up into subplots for
# each month:
grid.map(sns.lineplot, 'Day', 'Mean_Duration', color=mean_line_color_3_3_2)
grid.map(sns.lineplot, 'Day', 'Max_Duration', color=max_line_color_3_3_2)
grid.map(sns.lineplot, 'Day', 'Min_Duration', color=min_line_color_3_3_2)
grid.map(sns.lineplot, 'Day', 'Count_Durations', color=count_line_color_3_3_2)

axes = grid.axes.flatten()

grid.set_ylabels('Request Data', fontsize=15, labelpad=15)
grid.set_xlabels('Day', fontsize=15, labelpad=15)

place_month_names_on_axes(is_single_axes=False, axes_arr=axes, loc_str='left', font_size=17)

grid.set(yscale='symlog')  # Retain zero values
# ax.set_ylim(bottom=-0.25)  # 0 doesn't work...

grid.set_titles(fontsize=200)

# grid.set_xticklabels('')

grid.fig.subplots_adjust(top=0.85, left=0.125, right=0.997, hspace=0.2)  # These are magic numbers, do not adjust.
grid.fig.suptitle('Daily Max, Min and Mean of Req. Durations and Req. Load (3 mo.)', fontsize=20)

grid.fig.legend(handles=(axes[0].lines[3], axes[0].lines[2], axes[0].lines[1], axes[0].lines[0]), labels=('Request Load/Count', 'Min Req. Duration', 'Max Req. Duration', 'Mean Req. Duration'), borderpad=0.5, labelspacing=0.8, fontsize=13, loc='upper right')

# grid.axes[i].tick_params(axis='both', which='major', labelsize=15)
# ax.set_xticklabels(longform_dur_stats_3_mon_by_day['Day_Label'])

# Settings that need to be applied to each axis:
for i in range(len(axes)):
    axes[i].tick_params(axis='both', which='major', labelsize=14)

# Last call:
# grid.fig.tight_layout()

# ax.legend(borderpad=1, labelspacing=1, fontsize=18)

if (clear_df_from_mem_3_3_2):
    del daily_request_stats_3_mo
```



For swarmplot (buggy)

```
[ ]: longform_dur_stats_3_mon_by_day = pd.melt(dur_stats_3_mon_by_day,␣
      ↪id_vars='Day', value_vars=['Min_Duration', 'Max_Duration', 'Mean_Duration',␣
      ↪'Count_Durations'], var_name='Dur_Stat_Type', value_name='Dur_Stat_Value')
     # longform_dur_stats_3_mon_by_day.set_index(keys='Day', inplace=True) #␣
      ↪duplicate keys OK!

     # Get Date and Weekday from hist_3_mon:
     # Is there a better way to get the time stamp?
     # hist_3_mon_day_index = history_3_mon.set_index(keys='Day')
     # history_3_mon
     # hist_3_mon_day_index = hist_3_mon_day_index.groupby('Day').max()
     # hist_3_mon_day_index['StartTimeStamp'] =␣
      ↪hist_3_mon_day_index['StartTimeStamp'].astype(str).str.split(' ',␣
      ↪expand=True)[0]
```

```
# hist_3_mon_day_index = hist_3_mon_day_index[['StartTimeStamp', 'Weekday',␣
 ↪'Month']]
# hist_3_mon_day_index.head()
# # longform_dur_stats_3_mon_by_day
```

Put data from above into it's own DF for swarmplot

[170]:
```
longform_dur_stats_3_mon_by_day = pd.merge(longform_dur_stats_3_mon_by_day,␣
 ↪hist_3_mon_day_index, how='left', on='Day')
```

[ ]:
```
longform_dur_stats_3_mon_by_day['Day_Label'] =␣
 ↪longform_dur_stats_3_mon_by_day['StartTimeStamp'].astype(str) + ' \n' +␣
 ↪longform_dur_stats_3_mon_by_day['Weekday'].astype(str)
longform_dur_stats_3_mon_by_day.set_index(keys='Day', inplace=True)
longform_dur_stats_3_mon_by_day.drop(axis=1, columns=['StartTimeStamp',␣
 ↪'Weekday'], inplace=True)
```

[ ]:
```
longform_dur_stats_3_mon_by_day.head()
```

Swarmplot of stats by day

Break up into 3 months..?

[172]:
```
import seaborn as sns
```

[ ]:
```
sns.set_style('darkgrid')

fig, ax = plt.subplots(figsize=(20,10))

colors = sns.color_palette(('#89D328', '#D30702', '#1299E2', '#1F2C28'))  #␣
 ↪MIN, MAX, MEAN, COUNT

sns.swarmplot(x=longform_dur_stats_3_mon_by_day.index, y='Dur_Stat_Value',␣
 ↪hue='Dur_Stat_Type', data=longform_dur_stats_3_mon_by_day, dodge=True,␣
 ↪edgecolor='gray', linewidth=.4, palette=colors, s=7, ax=ax)
# sns.regplot(x='Day', y='Mean_Duration', data=dur_stats_3_mon_by_day,␣
 ↪color='#1299E2', scatter_kws={'s': 0}, ax=ax)

ax.set_yscale('symlog')  # Retain zero values
ax.set_ylim(bottom=-0.25)  # 0 doesn't work...

ax.set_ylabel('Request Duration Stats', fontsize=20)
ax.set_xlabel('Day', fontsize=20)
ax.set_title('Min, Max, Mean, and Count of Request Durations by Week (3 mo.)',␣
 ↪loc='center', fontsize=25)

ax.tick_params(axis='both', which='major', labelsize=15)
ax.set_xticklabels(longform_dur_stats_3_mon_by_day['Day_Label'])

ax.legend(borderpad=1, labelspacing=1, fontsize=18)
```

4.4 4. Find relationships between min/max/mean/count of request durations per day with a "correlogram"

[95]:
```
# Add a function that pulls the data which allows the user to filter data IF␣
 ↪WANTED (make a
# default that pulls all historical data?).
```

Drop excess fields we don't need here:

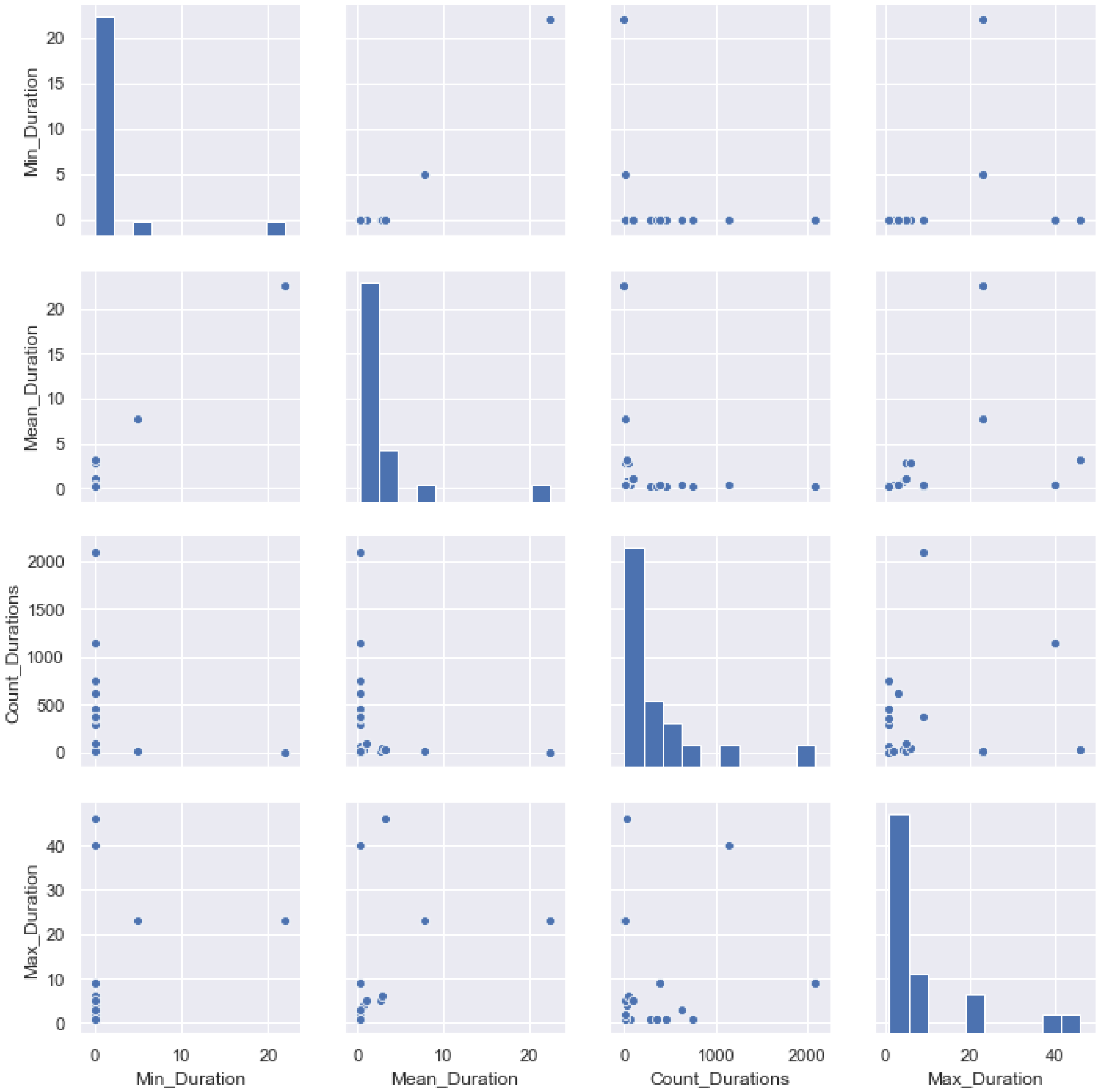
[85]:
```
# Basis DataFrame:
dur_stats_only_3_mo_day = dur_stats_3_mo_day.drop(axis=1, columns=['Month'])
dur_stats_only_3_mo_day.head()
```

[85]:
```
     Min_Duration  Mean_Duration  Count_Durations  Max_Duration
Day
2.0           0.0       0.264368             2088           9.0
3.0           0.0       0.330986             1136          40.0
4.0           0.0       0.252212              452           1.0
5.0           0.0       0.232639              288           1.0
6.0           0.0       0.252078              361           1.0
```

[ ]:
```
# longform_dur_stats_3_mon_day = pd.melt(dur_stats_3_mon_by_day.reset_index(),␣
 ↪value_vars=['Min_Duration', 'Max_Duration', 'Mean_Duration',␣
 ↪'Count_Durations'], var_name='Dur_Stat_Type', value_name='Dur_Stat_Value')
# longform_dur_stats_3_mon_day.dropna(inplace=True)
# longform_dur_stats_3_mon_day.head(2)
```

[89]:
```
sns.pairplot(data=dur_stats_only_3_mo_day, palette='husl')
```

[89]:
```
<seaborn.axisgrid.PairGrid at 0x13df8e90>
```



4.5 5. Total health history

5.1 - Total health history stats by month This section is under construction. My goal is to aggregate all the historical data by month in SQL and then make a figure with a subplot/Axes for each year and show all historical stats per month over the whole history of the DB.

I just have to separate year and month from being in the same column in the DF...

[179]:

```
sql = ('SELECT '
            ' MAX(YEAR([StartTimeStamp]))'
            ' ,MONTH(StartTimeStamp) as Month'
            ' ,COUNT(datediff(second, [StartTimeStamp], [EndTimeStamp])) as␣
↪Count_Durations'
```

```
            ' ,MAX(datediff(second, [StartTimeStamp], [EndTimeStamp])) as␣
 ↪Max_Duration'
            ' ,MIN(datediff(second, [StartTimeStamp], [EndTimeStamp])) as␣
 ↪Min_Duration'
            ' ,AVG(datediff(second, [StartTimeStamp], [EndTimeStamp])) as␣
 ↪Mean_Duration'
        ' FROM [test].[Request]'
        ' WHERE [StartTimeStamp] < [EndTimeStamp]'
        ' GROUP BY MONTH(StartTimeStamp), YEAR(StartTimeStamp)'
        ' UNION ALL'
        ' SELECT'
           ' MAX(YEAR([StartTimeStamp]))'
            ' ,MONTH(StartTimeStamp) as Month'
            ' ,COUNT(datediff(second, [StartTimeStamp], [EndTimeStamp])) as␣
 ↪Count_Durations'
            ' ,MAX(datediff(second, [StartTimeStamp], [EndTimeStamp])) as␣
 ↪Max_Duration'
            ' ,MIN(datediff(second, [StartTimeStamp], [EndTimeStamp])) as␣
 ↪Min_Duration'
            ' ,AVG(datediff(second, [StartTimeStamp], [EndTimeStamp])) as␣
 ↪Mean_Duration'
        ' FROM [test].[Request_History]'
        ' WHERE [StartTimeStamp] < [EndTimeStamp]'
        ' GROUP BY MONTH(StartTimeStamp), YEAR(StartTimeStamp)'
    )

monthly_requests_hist = pd.read_sql(sql, reporting_conn, index_col='Month')
monthly_requests_hist.reset_index(inplace=True)
monthly_requests_hist.sort_values(by=['Month'], inplace=True)
```

[180]:

```
monthly_requests_hist
```

[180]:

```
    Month        Count_Durations  Max_Duration  Min_Duration  Mean_Duration
39      1  2017            72662           387             0              2
28      1  2015            14698           400             0              4
27      1  2018            91809           560             0              2
72      1  2019            96776           581             0              1
69      1  2016            40380           438             0              2
..    …   …              …           …           …            …
12     12  2013             5065           202             0             10
65     12  2017            66244           501             0              2
23     12  2014             9120           624             0              6
53     12  2018            71129           332             0              1
50     12  2015            30787           306             0              2

[74 rows x 6 columns]
```

Below is code that does something similar, dut in pandas. It will probably not be used. Running this currently may cause memory errors.

[164]:
```
def prep_monthly_data_5_1(exclude_weekends: bool) -> pd.DataFrame:

    base_data_hist_w_dates = get_df_requests_with_date_cols_from_db(2015,2019)

    if (exclude_weekends):
        get_df_no_weekends(base_data_hist_w_dates, 'Weekday')

    # Aggregate Duration for min, max and count over each week (using named␣
 ↪aggregation, which drops all non-aggregated, non-index cols):
    month_aggs = base_data_hist_w_dates.groupby(pd.Grouper(freq='M')).agg(
    Min_Duration=pd.NamedAgg(column='Duration', aggfunc='min'),
    Max_Duration=pd.NamedAgg(column='Duration', aggfunc='max'),
    Mean_Duration=pd.NamedAgg(column='Duration', aggfunc=np.mean),
    Count_Durations=pd.NamedAgg(column='Duration', aggfunc='count'),
    First_Day=pd.NamedAgg(column='Day', aggfunc='min'),
    Last_Day=pd.NamedAgg(column='Day', aggfunc='max'),
    Month=pd.NamedAgg(column='Month', aggfunc='max'),
    Year=pd.NamedAgg(column='Year', aggfunc='max')
    #     Weekday=pd.NamedAgg(column='Duration', aggfunc=np.first)
    )
    month_aggs.dropna(inplace=True)

    # Add the Month_Label for labeling the plot:
    month_aggs['Month_Label'] = month_aggs['Year'].astype(int).astype(str) + '/
 ↪' + month_aggs['Month'].astype(int).astype(str) + '\ndays: ' +␣
 ↪month_aggs['First_Day'].astype(int).astype(str) + '-' +␣
 ↪month_aggs['Last_Day'].astype(int).astype(str)
    #     month_aggs_historical['Year'] = month_aggs_historical['Year'].
 ↪astype(str)[:4]
    # Drop columns that we only needed temporarily, to create the Week_Label␣
 ↪above:
    month_aggs.drop(axis=1, columns=['First_Day', 'Last_Day'], inplace=True)

    return month_aggs
```

[ ]:
```
# Bring in the data:
month_aggs_historical = prep_monthly_data_5_1(
    exclude_weekends=True
)
```

[ ]:
```
month_aggs_historical.head()
```

5.1.1 - Plot of all historical stats aggregated MONTHLY, with subplots/Axes for each year, all data This code will be used with the data loaded directly from SQL above, once bugs are fixed...

Figure definitions:

[61]:

```
marker_size_5_1_1 = 100

mean_line_color_5_1_1 = '#ffaa00'
min_line_color_5_1_1 = '#54b536'
max_line_color_5_1_1 = '#bb311b'
count_line_color_5_1_1 = '#69c7ff'
```

[ ]:

```
mon_hist_grid = sns.FacetGrid(row='Year', data=monthly_requests_hist, height=3.
 ↪6, aspect=3.7, legend_out=True)

mon_hist_grid.map(sns.lineplot, 'Month', 'Mean_Duration',␣
 ↪color=mean_line_color_5_1_1, estimator=None)
mon_hist_grid.map(sns.scatterplot, 'Month', 'Mean_Duration',␣
 ↪color=mean_line_color_5_1_1, s=marker_size_5_1_1)

mon_hist_grid.map(sns.lineplot, 'Month', 'Min_Duration',␣
 ↪color=min_line_color_5_1_1, estimator=None)
mon_hist_grid.map(sns.scatterplot, 'Month', 'Min_Duration',␣
 ↪color=min_line_color_5_1_1, s=marker_size_5_1_1)

mon_hist_grid.map(sns.lineplot, 'Month', 'Max_Duration',␣
 ↪color=max_line_color_5_1_1, estimator=None)
mon_hist_grid.map(sns.scatterplot, 'Month', 'Max_Duration',␣
 ↪color=max_line_color_5_1_1, s=marker_size_5_1_1)

mon_hist_grid.map(sns.lineplot, 'Month', 'Count_Durations',␣
 ↪color=count_line_color_5_1_1, estimator=None)
mon_hist_grid.map(sns.scatterplot, 'Month', 'Count_Durations',␣
 ↪color=count_line_color_5_1_1, s=marker_size_5_1_1)

axes = mon_hist_grid.axes.flatten()

mon_hist_grid.set_ylabels('Request Duration Stats', fontsize=15, labelpad=15)
mon_hist_grid.set_xlabels('Month', fontsize=15, labelpad=15)

set_year_name_axes_titles(axes, monthly_requests_hist, 'left', 'Weekday')

mon_hist_grid.set(yscale='symlog')  # Retain zero values

mon_hist_grid.set_titles(fontsize=200)
```

```
# grid.set_xticklabels('')

mon_hist_grid.fig.subplots_adjust(top=0.8, left=0.125, right=0.997, hspace=0.2)␣
 ↪ # These are magic numbers, do not adjust.
mon_hist_grid.fig.suptitle('Mean, Min, Max and Count of Request Durations by␣
 ↪Month (to-Date)', fontsize=20)

mon_hist_grid.fig.legend(handles=(axes[0].lines[3], axes[0].lines[2], axes[0].
 ↪lines[1], axes[0].lines[0]), labels=('Request Load/Count', 'Max Req.␣
 ↪Duration', 'Min Req. Duration', 'Mean Req. Duration'), borderpad=0.5,␣
 ↪labelspacing=0.8, fontsize=13, loc='upper right')

# grid.axes[i].tick_params(axis='both', which='major', labelsize=15)

# Settings that need to be applied to each axis:
for i in range(len(axes)):
    axes[i].set_xticklabels(monthly_requests_hist['Month_Label'], rotation=45)
    axes[i].tick_params(axis='both', which='major', labelsize=14)
    axes[i].set_ylim(bottom=-0.5)  # 0 doesn't work...
```

5.2 - Total health history stats by week The following code was also done using all historical data in a pandas DF, so it will not be used as it takes a ton of memory.

[25]:

```
import seaborn as sns
```

[182]:

```
def prep_weekly_data_5_2(exclude_weekends: bool) -> pd.DataFrame:

    base_data_hist_w_dates = get_df_requests_with_date_cols_from_db(2015,2019)

    if (exclude_weekends):
        weekday_filter = ( (base_data_hist_w_dates['Weekday'] != 'Saturday') &␣
 ↪(base_data_hist_w_dates['Weekday'] != 'Sunday') )
        base_data_hist_w_dates.where(weekday_filter, inplace=True)
        base_data_hist_w_dates.dropna(inplace=True)

    # Aggregate Duration for min, max and count over each week (using named␣
 ↪aggregation, which drops all non-aggregated, non-index cols):
    week_aggs = base_data_hist_w_dates.groupby(pd.Grouper(freq='W')).agg(
    Min_Duration=pd.NamedAgg(column='Duration', aggfunc='min'),
    Max_Duration=pd.NamedAgg(column='Duration', aggfunc='max'),
    Mean_Duration=pd.NamedAgg(column='Duration', aggfunc=np.mean),
    Count_Durations=pd.NamedAgg(column='Duration', aggfunc='count'),
    First_Day=pd.NamedAgg(column='Day', aggfunc='min'),
    Last_Day=pd.NamedAgg(column='Day', aggfunc='max'),
    Week=pd.NamedAgg(column='Week', aggfunc='max'),
    Month=pd.NamedAgg(column='Month', aggfunc='max'),
    Year=pd.NamedAgg(column='Year', aggfunc='max')
    #     Weekday=pd.NamedAgg(column='Duration', aggfunc=np.first)
    )
    week_aggs.dropna(inplace=True)

    # Add the Month_Label for labeling the plot:
    week_aggs['Week_Label'] = week_aggs['Year'].astype(int).astype(str) + '/' +␣
↪week_aggs['Month'].astype(int).astype(str) + '\ndays: ' +␣
↪week_aggs['First_Day'].astype(int).astype(str) + '-' + week_aggs['Last_Day'].
↪astype(int).astype(str)
    #     month_aggs_historical['Year'] = month_aggs_historical['Year'].
↪astype(str)[:4]
    # Drop columns that we only needed temporarily, to create the Week_Label␣
↪above:
    week_aggs.drop(axis=1, columns=['First_Day', 'Last_Day'], inplace=True)

    return week_aggs
```

[ ]:

```
# RUN THIS CELL TO RESET DATA:
wk_aggs_historical = prep_weekly_data_5_2(
    exclude_weekends=True
)
```

[42]:

```
wk_aggs_historical.head()
```

[42]:

```
                Min_Duration  Max_Duration  Mean_Duration  Count_Durations  \
StartTimeStamp
2015-01-25               0.0           4.0       0.625000               32
2015-02-01               0.0          46.0       0.388060             1474
2015-02-08               0.0          40.0       0.277457             4325
2015-02-15               0.0          23.0       0.433790              438
2015-03-01              23.0          23.0      23.000000                1

                Week  Month    Year         Week_Label
StartTimeStamp
2015-01-25       4.0    1.0  2015.0  2015/1\ndays: 19-22
2015-02-01       5.0    1.0  2015.0  2015/1\ndays: 26-30
2015-02-08       6.0    2.0  2015.0    2015/2\ndays: 2-6
2015-02-15       7.0    2.0  2015.0   2015/2\ndays: 9-11
2015-03-01       9.0    2.0  2015.0  2015/2\ndays: 23-23
```

5.2.1 - Plot of weekly stats, historical DO NOT RUN the figure source code below if you want to keep the figure. The gives a picture of what we might want to do with the data in SQL to make a similar visualization of historical data by week, but the current code is buggy.

Plot definitions:

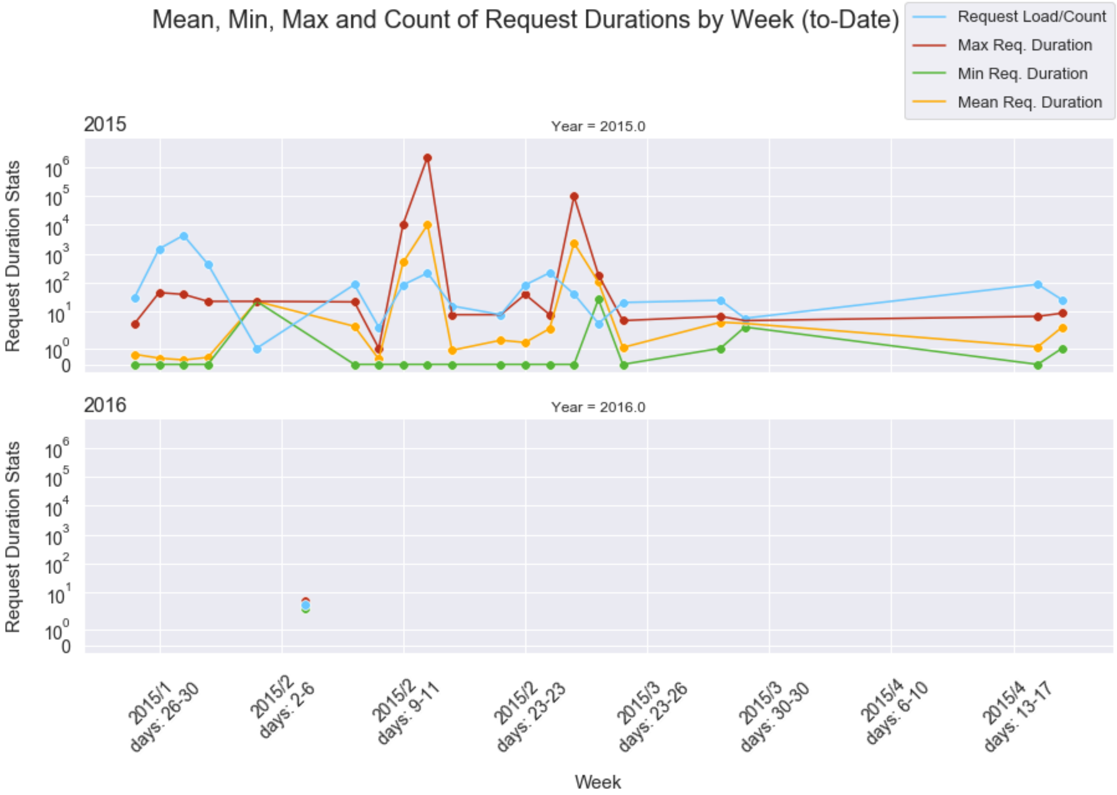
[57]:

```
marker_size_5_2_1 = 60

mean_line_color_5_2_1 = '#ffaa00'
min_line_color_5_2_1 = '#54b536'
max_line_color_5_2_1 = '#bb311b'
count_line_color_5_2_1 = '#69c7ff'
```

[60]:

```
sns.set(style='darkgrid')

wk_hist_grid = sns.FacetGrid(row='Year', data=wk_aggs_historical, height=4,␣
 ↪aspect=3.3, legend_out=True)

wk_hist_grid.map(sns.lineplot, 'Week', 'Mean_Duration',␣
 ↪color=mean_line_color_5_2_1, estimator=None)
wk_hist_grid.map(sns.scatterplot, 'Week', 'Mean_Duration',␣
 ↪color=mean_line_color_5_2_1, s=marker_size_5_2_1)

wk_hist_grid.map(sns.lineplot, 'Week', 'Min_Duration',␣
 ↪color=min_line_color_5_2_1, estimator=None)
wk_hist_grid.map(sns.scatterplot, 'Week', 'Min_Duration',␣
 ↪color=min_line_color_5_2_1, s=marker_size_5_2_1)

wk_hist_grid.map(sns.lineplot, 'Week', 'Max_Duration',␣
 ↪color=max_line_color_5_2_1, estimator=None)
wk_hist_grid.map(sns.scatterplot, 'Week', 'Max_Duration',␣
 ↪color=max_line_color_5_2_1, s=marker_size_5_2_1)

wk_hist_grid.map(sns.lineplot, 'Week', 'Count_Durations',␣
 ↪color=count_line_color_5_2_1, estimator=None)
wk_hist_grid.map(sns.scatterplot, 'Week', 'Count_Durations',␣
 ↪color=count_line_color_5_2_1, s=marker_size_5_2_1)

axes = wk_hist_grid.axes.flatten()

wk_hist_grid.set_ylabels('Request Duration Stats', fontsize=15, labelpad=15)
wk_hist_grid.set_xlabels('Week', fontsize=15, labelpad=15)

set_year_name_axes_titles(axes, wk_aggs_historical, 'left', 'Weekday')

wk_hist_grid.set(yscale='symlog')  # Retain zero values

wk_hist_grid.set_titles(fontsize=200)

# grid.set_xticklabels('')
```

```
wk_hist_grid.fig.subplots_adjust(top=0.8, left=0.125, right=0.997, hspace=0.2) ␣
 ↪# These are magic numbers, do not adjust.
wk_hist_grid.fig.suptitle('Mean, Min, Max and Count of Request Durations by␣
 ↪Week (to-Date)', fontsize=20)

wk_hist_grid.fig.legend(handles=(axes[0].lines[3], axes[0].lines[2], axes[0].
 ↪lines[1], axes[0].lines[0]), labels=('Request Load/Count', 'Max Req.␣
 ↪Duration', 'Min Req. Duration', 'Mean Req. Duration'), borderpad=0.5,␣
 ↪labelspacing=0.8, fontsize=13, loc='upper right')

# grid.axes[i].tick_params(axis='both', which='major', labelsize=15)

# Settings that need to be applied to each axis:
for i in range(len(axes)):
    axes[i].set_xticklabels(wk_aggs_historical['Week_Label'], rotation=45)
    axes[i].tick_params(axis='both', which='major', labelsize=14)
    axes[i].set_ylim(bottom=-0.5)  # 0 doesn't work...
```


Mean, Min, Max and Count of Request Durations by Week (to-Date)
Request Load/Count
Max Req. Duration
Min Req. Duration
Mean Req. Duration
Year = 2015.0
Request Duration Stats
10^6
10^5
10^4
10^3
10^2
10^1
10^0
Year = 2016.0
Request Duration Stats
10^6
10^5
10^4
10^3
10^2
10^1
10^0
2015/1 days: 26-30
2015/2 days: 2-6
2015/2 days: 9-11
2015/2 days: 23-23
2015/3 days: 23-26
2015/3 days: 30-30
2015/4 days: 6-10
2015/4 days: 13-17
Week