Modeling Techniques in Predictive Analytics with Python and R

A Guide to Data Science

THOMAS W. MILLER

Associate Publisher: Amy Neidlinger
Executive Editor: Jeanne Glasser
Operations Specialist: Jodi Kemper
Cover Designer: Alan Clements
Managing Editor: Kristy Hart
Project Editor: Andy Beaster
Senior Compositor: Gloria Schurick
Manufacturing Buyer: Dan Uhrig

Published by Pearson Education, Inc.
Upper Saddle River, New Jersey 07458

Printed in the United States of America

First Printing October 2014

ISBN-10: 0-13-3892069
ISBN-13: 978-0-13-389206-2

Pearson Education LTD.
Pearson Education Australia PTY, Limited.
Pearson Education Singapore, Pte. Ltd.
Pearson Education Asia, Ltd.
Pearson Education Canada, Ltd.
Pearson Educacin de Mexico, S.A. de C.V.
Pearson Education—Japan
Pearson Education Malaysia, Pte. Ltd.
Library of Congress Control Number: 2014948913

6 Operations Management

"Go ahead, make my day."

—CLINT EASTWOOD AS HARRY CALLAHAN IN *Sudden Impact* (1983)

I have something in common with Michael Feldman of the public radio show *Whad'Ya Know.* For a few years I worked with Union Cab Cooperative of Madison, Wisconsin. Union Cab, an employee-owned and operated cooperative, has years of data showing when and where they pick up and drop off passengers. The precise path of travel for every ride is tracked by GPS sensors and logged with time stamps every few seconds. With an active call center, a sixty-cab fleet, and an operation that never shuts down, Union Cab is a treasure trove of data. Working with these data gave me an appreciation for the importance of analytics in operations management.

Operations managers benefit by using the tools of operations research, including queueing theory, mathematical programming, and process and discrete event simulation. We will touch on the first two of these areas using public domain data from the call center of "Anonymous Bank" in Israel.[1] We focus on data from February 1999. We begin, as we often do, with data visualization, looking at wait times for service. Then we use a queueing model to estimate workforce requirements and mathematical programming to define an optimal workforce schedule.

[1] Call center data from "Anonymous Bank" were provided by Avi Mandelbaum, with the help of Ilan Guedj. These data have been used in other published works, including Brown et al. (2005).

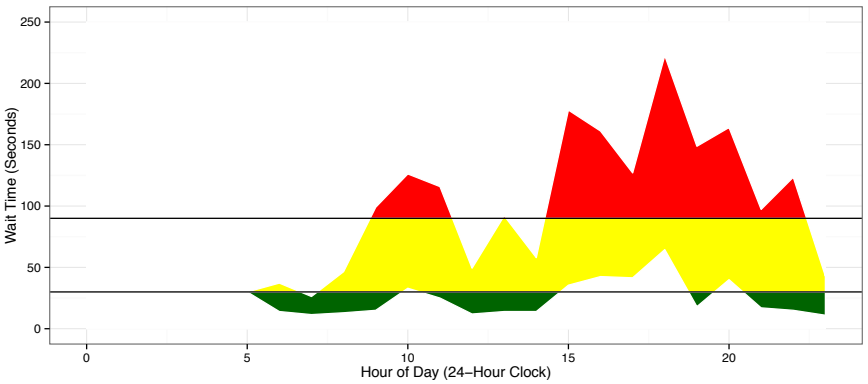
Exhibits 6.1 through 6.6 include wait-time ribbon plots for the days that the bank was open during the first week of February 1999. Each wait-time ribbon provides a visualization of twenty-four wait-time distributions, one for each hour of the day. The bottom of a ribbon represents the 50^{th} percentile or median of wait times during any given hour. Fifty percent of calls fall below the bottom of the ribbon. The top of the ribbon represents the 90^{th} percentile of wait times, so ten percent of calls fall above the top of the ribbon. Tabled values below the ribbons provide additional documentation for operations managers, displaying numbers of service operators, total calls per hour, calls served, and calls dropped (abandoned).

Call center performance is judged relative to management goals, which are portrayed as horizontal lines on each wait-time ribbon. Suppose that one of the bank's service goals is to realize wait times of thirty seconds or less, with wait times in excess of ninety seconds being considered intolerable. For each day's data, we can see that large segments of many ribbons fall above the ninety-second line. This means that many callers are waiting for more than ninety seconds before being served.

The wait-time ribbons suggest that the bank may want to schedule additional call center staff to meet its service performance goals. Workforce scheduling involves two modeling tasks: estimating workforce needs and scheduling workers to meet those needs. There can be wide variability in requirements from one hour to the next, with bank customers calling during their lunch hour or afternoon break times. To match call center traffic, the bank may benefit by having flexible shifts or shifts with various start times.

Given wide variability in traffic, estimating workforce requirements for a call center is a challenging analytic problem. Alternative methods include call-level forecasting models, queueing models, process simulation, or some combination of these. With data from the first week of February 1999, we can estimate arrival rates and service rates for any day of the week. We use data from Wednesdays in February to demonstrate the process. Call arrival and service rates are shown in figure 6.7.

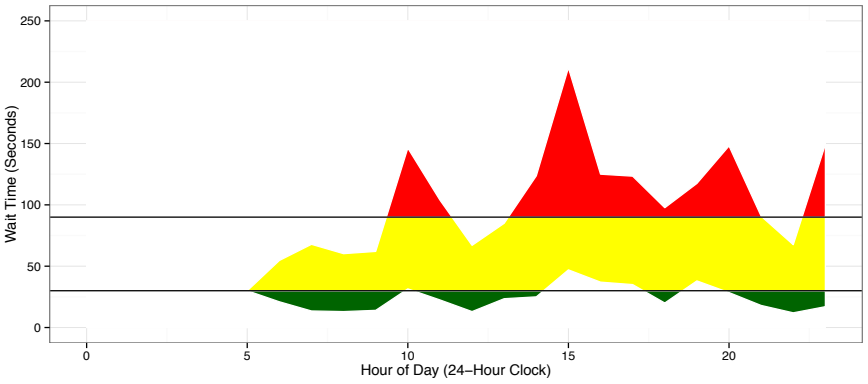
Figure 6.1. *Call Center Operations for Monday*



Hour:	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Servers:	0	0	0	0	0	0	1	5	8	8	9	10	10	10	9	10	6	5	7	5	5	4	4	4
Calls:	4	3	1	0	0	1	9	50	134	118	110	127	92	106	109	96	88	68	120	69	87	61	49	45
Served:	0	0	0	0	0	0	1	44	121	102	93	109	89	95	102	80	74	56	93	55	73	54	44	44
Dropped:	4	3	1	0	0	1	8	6	13	16	17	18	3	11	7	16	14	12	27	14	14	7	5	1

Bottom of ribbon = 50th percentile of wait times Top of ribbon = 90th percentile of wait times.

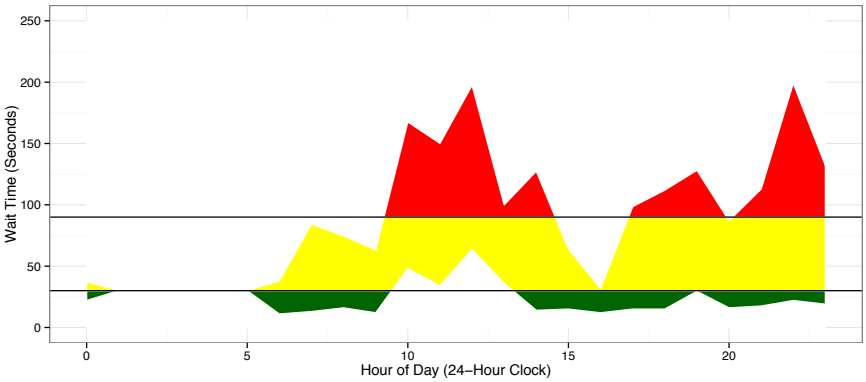
Figure 6.2. *Call Center Operations for Tuesday*



Hour:	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Servers:	0	0	0	0	0	0	1	6	9	9	9	9	9	11	10	11	9	8	8	7	7	6	5	3
Calls:	2	2	1	0	0	4	9	58	124	131	158	130	114	132	133	147	157	115	81	89	66	75	51	55
Served:	0	0	0	0	0	0	1	52	111	123	140	108	106	116	109	105	130	95	75	80	61	66	49	49
Dropped:	2	2	1	0	0	4	8	6	13	8	18	22	8	16	24	42	27	20	6	9	5	9	2	6

Bottom of ribbon = 50th percentile of wait times Top of ribbon = 90th percentile of wait times.

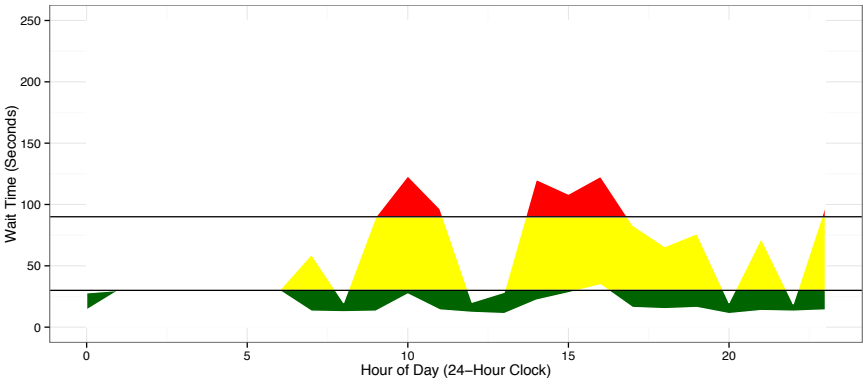
Figure 6.3. *Call Center Operations for Wednesday*



Hour:	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Servers:	0	0	0	0	0	0	1	6	7	9	9	9	9	10	11	11	8	7	7	7	7	6	3	3
Calls:	7	2	2	0	1	1	5	41	119	104	125	125	102	494	145	117	64	68	83	104	66	68	49	46
Served:	0	0	0	0	0	0	1	36	100	91	103	94	61	75	134	106	60	63	67	95	62	62	41	42
Dropped:	7	2	2	0	1	1	4	5	19	13	22	31	41	419	11	11	4	5	16	9	4	6	8	4

Bottom of ribbon = 50th percentile of wait times Top of ribbon = 90th percentile of wait times.

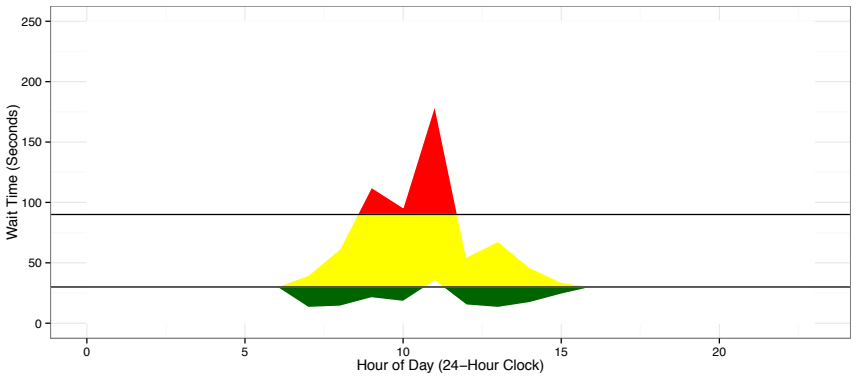
Figure 6.4. *Call Center Operations for Thursday*



Hour:	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Servers:	0	0	0	0	0	0	0	4	8	8	9	10	10	10	8	10	8	6	7	8	7	6	5	5
Calls:	6	0	0	0	0	1	2	32	66	95	117	91	83	87	95	105	152	106	81	70	62	34	43	36
Served:	0	0	0	0	0	0	0	28	64	89	106	82	80	85	88	89	117	97	77	67	55	30	43	33
Dropped:	6	0	0	0	0	1	2	4	2	6	11	9	3	2	7	16	35	9	4	3	7	4	0	3

Bottom of ribbon = 50th percentile of wait times Top of ribbon = 90th percentile of wait times.

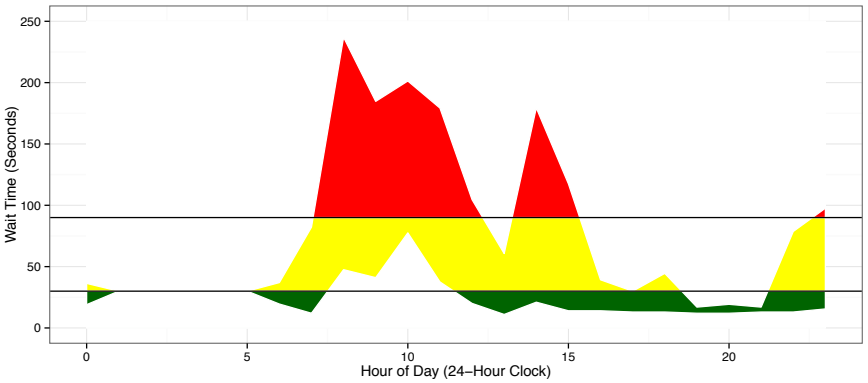
Figure 6.5. *Call Center Operations for Friday*



Hour:	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Servers:	0	0	0	0	0	0	1	5	6	7	6	7	7	5	0	0	0	0	0	0	0	0	0	0
Calls:	3	0	4	1	1	1	2	28	79	68	75	93	77	48	22	7	4	0	0	2	1	0	0	0
Served:	0	0	0	0	0	0	1	27	71	66	66	75	71	47	0	0	0	0	0	0	0	0	0	0
Dropped:	3	0	4	1	1	1	1	1	8	2	9	18	6	1	22	7	4	0	0	2	1	0	0	0

Bottom of ribbon = 50th percentile of wait times Top of ribbon = 90th percentile of wait times.

Figure 6.6. *Call Center Operations for Sunday*



Hour:	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Servers:	0	0	0	0	0	0	0	3	7	7	7	8	9	10	12	14	12	10	9	9	9	9	6	5
Calls:	7	2	1	1	0	2	6	34	124	129	128	130	102	83	109	108	155	103	64	43	50	62	47	42
Served:	0	0	0	0	0	0	0	29	88	94	92	95	91	74	95	97	145	93	60	41	49	60	43	35
Dropped:	7	2	1	1	0	2	6	5	36	35	36	35	11	9	14	11	10	10	4	2	1	2	4	7

Bottom of ribbon = 50th percentile of wait times Top of ribbon = 90th percentile of wait times.

Figure 6.7. *Call Center Arrival and Service Rates on Wednesdays*

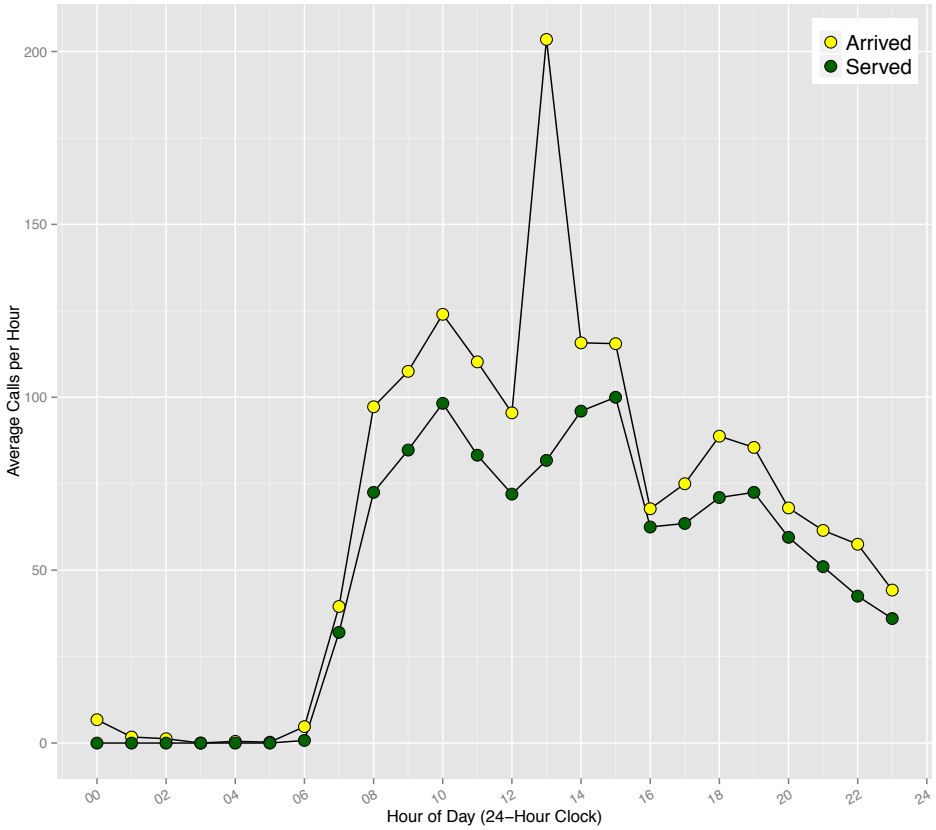


Table 6.1. *Call Center Shifts and Needs for Wednesdays*

Hour	StartTime	Shift1	Shift2	Shift3	Shift4	Shift5	Shift6	Shift7	Shift8	Need
"00"	Midnight	1	0	0	0	0	0	0	0	0
"01"	1am	1	0	0	0	0	0	0	0	0
"02"	2am	1	0	0	0	0	0	0	0	0
"03"	3am	1	0	0	0	0	0	0	0	0
"04"	4am	1	0	0	0	0	0	0	0	0
"05"	5am	1	0	0	0	0	0	0	0	0
"06"	6am	0	1	0	0	0	0	0	0	1
"07"	7am	0	1	0	0	0	0	0	0	4
"08"	8am	0	1	1	0	0	0	0	0	8
"09"	9am	0	1	1	0	0	0	0	0	9
"10"	10am	0	1	1	1	0	0	0	0	10
"11"	11am	0	1	1	1	0	0	0	0	9
"12"	Noon	0	0	1	1	1	0	0	0	8
"13"	1pm	0	0	1	1	1	0	0	0	16
"14"	2pm	0	0	0	1	1	1	0	0	10
"15"	3pm	0	0	0	1	1	1	0	0	10
"16"	4pm	0	0	0	0	1	1	1	0	6
"17"	5pm	0	0	0	0	1	1	1	0	7
"18"	6pm	0	0	0	0	0	1	1	1	8
"19"	7pm	0	0	0	0	0	1	1	1	8
"20"	8pm	0	0	0	0	0	0	1	1	6
"21"	9pm	0	0	0	0	0	0	1	1	6
"22"	10pm	0	0	0	0	0	0	0	1	5
"23"	11pm	0	0	0	0	0	0	0	1	4

For this example, we apply standard queueing models to estimate the workforce requirements for each hour of the day on Wednesdays. Let us assume that the bank wants no more than 50 percent of callers to wait in queue before speaking with a service agent. The bank can benefit from having flexible shift start times. Let us assume that the bank's call center shifts are six hours in duration, beginning at specific even-numbered hours of the day and evening. In particular, suppose that there is a midnight-to-6 a.m. shift and shifts beginning every two hours from 6 a.m. through 6 p.m. Workforce shifts are shown in table 6.1, with one column for each shift. Binary indicators in cells for each shift column provide a representation of the shift for input to mathematical programming. The number 1 indicates that an hour is part of a shift, and the number 0 indicates that an hour is not part of a shift. Workforce hourly needs are also shown in the table. These were estimated from a queueing model.[2]

[2] We used a standard Erlang C model to estimate the number of service operators for each hour of the day on Wednesdays. "Anonymous Bank" case data suggest that workers can handle an average of fifteen calls an hour. Many organizations utilize the Erlang C model for workforce scheduling. The Erlang C model assumes that calls are random Poisson arrivals. The model does not take abandoned calls into account. Revisions of Erlang C have been proposed by statisticians and operations researchers (Brown et al. 2005; Janssen, Leeuwaarden, and Zwart 2011).

Table 6.2. *Call Center Problem and Solution*

Shift	Start Time	Duration (hours)	Hourly Cost	Shift Cost	Optimal Shift Schedule	Total Cost for Shift Schedule
1	0	6	42	252	0	0
2	6	6	48	288	4	1,152
3	8	6	30	180	8	1,440
4	10	6	30	180	4	720
5	12	6	30	180	4	720
6	2	6	48	288	2	576
7	4	6	48	288	1	288
8	6	6	48	288	5	1,440
			Total Minimum Daily Cost (ILS)			6,336

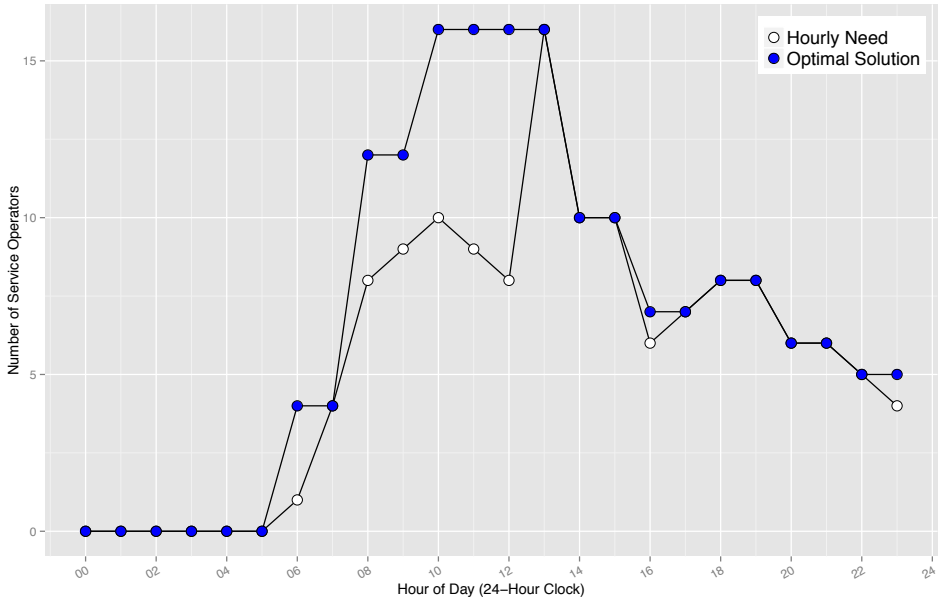
The business problem of workforce scheduling concerns scheduling workers in a way that satisfies resource needs while minimizing costs. This is a constrained optimization problem requiring an integer solution. We use integer programming, a type of mathematical programming, to obtain the optimal workforce schedule for the bank. Inputs to the program include results from the queueing model, information about workforce shifts, and salaries paid to workers in each shift. Table 6.2 and figure 6.8 show the optimal call center solution obtained through integer programming.[3]

[3] Mathematical programming methods for optimal shift scheduling date back to the work of Dantzig (1954). Let a_{ij} be 1 if shift type j includes hour i and be zero otherwise. Let c_j be the cost of shift type j. And let the decision variable x_j be the number of call center operators to be scheduled for shift type j. Hourly workloads are expressed in terms of the number of call center operators needed: b_j. We define a constrained optimization problem with an objective function to be minimized. For call center shift scheduling, we minimize total call operator costs subject to constraints defined by hourly needs:

$$\text{Minimize} \quad \sum_{ij} c_i x_j \quad \text{subject to} \quad \sum_{ij} a_{ij} x_j \geq b_i$$

Hourly worker needs b_i and operators to be scheduled for shifts x_j are constrained to be whole numbers, making call center scheduling an integer programming problem. For our example, suppose this is a 24-hour call center with six-hour shifts for workers. In particular, there are eight shift types, defined by start times at midnight, 6 a.m., 8 a.m., 10 a.m., noon, 2 p.m., 4 p.m., and 6 p.m. Suppose workers starting shifts at 8 a.m., 10 a.m., and noon make 30 ILS per hour (180 ILS shift cost). Workers starting shifts at 6 a.m., 2 p.m., 4 p.m., and 6 p.m. make 48 ILS per hour (288 ILS shift cost). And workers who start shifts at midnight make 42 ILS per hour (252 ILS shift cost). These data define the elements of a cost vector c_j. To determine the best schedule of call center shifts, we set up the matrix elements a_{ij} and the operators-needed-vector elements b_i. Then we use computer algorithms to solve the integer programming problem. Our objective is to minimize costs, so the best or optimal call center schedule will be the cost-minimizing schedule. The solution to this integer programming problem will show the number of shifts of each type x_j, number of operators available each hour of the day, and the total operator cost. (The conversion rate from Israeli shekels to United States dollars was 1 ILS = 3.61 USD in June 2013.)

Figure 6.8. *Call Center Needs and Optimal Workforce Schedule*



We could test what we are calling the optimal solution through *process simulation* or, more specifically, *discrete event simulation*. We imagine that callers arrive according to a random Poisson process and that service operators continue to provide services as they have in the past, serving approximately fifteen calls per minute. We run a computer program that behaves as callers and service operators have behaved in the past. The wait-time output from the program could then be displayed in wait-time ribbons for management review.

Another way to build confidence in a proposed solution and to explore alternative solutions is to vary inputs to the integer programming algorithm. That is, we test the utility or stability of the solution in the face of varying inputs for staffing needs. Within operations research and the mathematical programming literature, this is called *sensitivity testing.*

Discussion of workforce scheduling methods and call center management is provided by Ernst et al. (2004) and Aksin, Armony, and Mehrotra (2007). For statistical approaches to workforce scheduling that move beyond the standard Erlang C model, see Brown et al. (2005). General discussion of queueing theory is provided by Kleinrock (2009) and Gross, Shortle, Thompson, and Harris (2008). Relevant programming methods are provided by Canadilla (2014).

Mathematical programming is a set of constrained optimization methods with applications across many areas of business and economics. For further reading in the area of mathematical programming, see Williams (2013).

Sensitivity testing can be executed within a statistical simulation environment, such as provided by Alfons, Templ, and Filzmoser (2010). Discrete event simulation with multi-server queues is often used to explore alternative call center schedules.

The Python program in exhibit 6.1 shows how to read the data for "Anonymous Bank," as well as information for call center shifts. The program sets up an hours-by-shift matrix and uses a queueing theory model to estimate the operators-needed-vector for each hour of the day. To determine the optimal or cost-minimizing schedule, the program utilizes an integer programming algorithm. The solution, reported in tables and figures, shows the number of shifts of each type, the number of operators scheduled for each hour of the day, and the total operator cost for the day.

The corresponding R program is provided in exhibit 6.2. This program draws upon R packages provided by Grolemund and Wickham (2014), Wickham and Chang (2014), Canadilla (2014), and Berkelaar (2014). For the plotting of wait-time ribbons, we call upon programs in the *R Code and Utilities* appendix (exhibits D.6 and D.7).

Exhibit 6.1. *Call Center Scheduling (Python)*

```
# Workforce Scheduling for Anonymous Bank Call Center (Python)

# prepare for Python version 3x features and functions
from __future__ import division, print_function

# import packages for analysis and modeling
import pandas as pd  # data frame operations
import numpy as np  # arrays and math functions
import datetime
from rpy2.robjects import r  # interface from Python to R

# Erlang C queueing theory
# input c = number of servers (positive integer)
#       r = ratio of arrival rate over service rate
# output = probability of waiting in queue (min 0, max 1)
# adapted from Pedro Canadilla (2014) function
# C_erlang in the R queueing package
def erlang_C (c = 1, r = 0):
    if (c <= 0):
        return(1)
    if (r <= 0):
        return(0)
    c = int(c)
    tot = 1
    for i in range(c-1):
        i = i + 1
        tot = 1 + (tot * i * (1/r))
    return(max(0, min(1, (r * (1/tot)) / (c - (r * (1 - (1/tot)))))))

# focus upon February 1999
call_center_input_data = pd.read_table('data_anonymous_bank_february.txt')
# examine the structure of these data
print(call_center_input_data.head)

# delete PHANTOM calls
call_center_data = \
    call_center_input_data[call_center_input_data['outcome'] != 'PHANTOM']

# negative VRU times make no sense... drop these rows from data frame
call_center_data = call_center_data[call_center_data['vru_time'] >= 0]

# calculate wait time as sum of vru_time and q_time
call_center_data['wait_time'] = call_center_data['vru_time'] + \
    call_center_data['q_time']

# define date variable with apply and lambda function
call_center_data['date'] = \
    call_center_data['date']\
    .apply(lambda d: datetime.datetime.strptime(str(d), '%y%m%d'))
```

```
# define day of week as an integer 0 = Monday 6 = Sunday
call_center_data['day_of_week'] = \
    call_center_data['date'].apply(lambda d: d.weekday())
# use dictionary object for mapping day_of_week to string
day_of_week_to_string = {0 : 'Monday',
     1 : 'Tuesday',
     2 : 'Wednesday',
     3 : 'Thursday',
     4 : 'Friday',
     5 : 'Saturday',
     6 : 'Sunday'}
call_center_data['day_of_week'] = \
    call_center_data['day_of_week'].map(day_of_week_to_string)

# check structure and contents of the data frame
print(call_center_data.head)

# examine frequency of calls by day of week
print(call_center_data['day_of_week'].value_counts())

# identify the hour of entry into the system
call_center_data['vru_entry'] = \
    call_center_data['vru_entry']\
    .apply(lambda d: datetime.datetime.strptime(str(d), '%H:%M:%S'))
call_center_data['call_hour'] = \
    call_center_data['vru_entry'].apply(lambda d: d.hour)

# check frequency of calls in February by hour and day of week
# note that pandas alphabetizes on output
print(pd.crosstab(call_center_data['day_of_week'],\
    call_center_data['call_hour'], margins = False))

# create an ordered table for frequency of calls
table_data = call_center_data.ix[:,['day_of_week', 'call_hour']]
day_of_week_to_ordered_day_of_week = {'Monday' : '2_Monday',
     'Tuesday' : '3_Tuesday',
     'Wednesday' : '4_Wednesday',
     'Thursday' : '5_Thursday',
     'Friday' : '6_Friday',
     'Saturday' : '7_Saturday',
     'Sunday' : '1_Sunday'}

table_data['ordered_day_of_week'] = \
    table_data['day_of_week'].map(day_of_week_to_ordered_day_of_week)
print(pd.crosstab(table_data['ordered_day_of_week'],\
    table_data['call_hour'], margins = False))

# select first week of February 1999 for data visualization and analysis
# that week began on Monday, February 1 and ended on Sunday, February 7
selected_week = call_center_data[call_center_data['date'] <
    datetime.date(1999, 2, 8)]
print(selected_week.head)
```

```
# check frequency of calls in February by hour and day of week
# create an ordered table for frequency of calls
# for the first week in February 1999
table_data = selected_week.ix[:,['day_of_week', 'call_hour']]
day_of_week_to_ordered_day_of_week = {'Monday' : '2_Monday',
     'Tuesday' : '3_Tuesday',
     'Wednesday' : '4_Wednesday',
     'Thursday' : '5_Thursday',
     'Friday' : '6_Friday',
     'Saturday' : '7_Saturday',
     'Sunday' : '1_Sunday'}
table_data['ordered_day_of_week'] = \
    table_data['day_of_week'].map(day_of_week_to_ordered_day_of_week)
print(pd.crosstab(table_data['ordered_day_of_week'],\
    table_data['call_hour'], margins = False))

# wait-time ribbons were created with R ggplot2 software
# Python packages ggplot or rpy2 could be used for plotting

# select Wednesdays in February for the queueing model
wednesdays = call_center_data[call_center_data['day_of_week'] == \
    'Wednesday']
print(wednesdays.head)
# arrival rate as average number of calls into VRU per hour
arrived_for_hour = wednesdays['call_hour'].value_counts()
check_hourly_arrival_rate = arrived_for_hour/4  # four Wednesdays in February 1999
print(check_hourly_arrival_rate)
# organize hourly arrival rates according to 24-hour clock
hourly_arrival_rate = [6.75, 1.75, 1.25, 0.00, 0.50, 0.25,\
    4.75, 39.50, 97.25,107.50, 124.00,110.25, 95.50,\
    203.50, 115.75, 115.50, 67.75, 75.00, 88.75,\
    85.50, 68.00, 61.50, 57.50, 44.25]
# service times may vary hour-by-hour due to differences
# in service requests and individuals calling hour-by-hour
# begin by selecting calls that receive service
wednesdays_served = wednesdays[wednesdays['server'] != \
    'NO_SERVER']
print(wednesdays_served.head)

hourly_mean_service_time =\
    wednesdays_served.pivot_table('ser_time', cols = ['call_hour'],\
    aggfunc = 'mean', margins = False)
# hourly service rate given the current numbers of service operators
served_for_hour = wednesdays_served['call_hour'].value_counts()
print(served_for_hour)

# compute service rate noting that there are 3600 seconds in an hour
# adding 60 seconds to each mean service time for time between calls
# this 60 seconds is the wrap up time or time an service agent remains
# unavailable to answer a new call after a call has been completed
mean_hourly_service_rate = 3600/(hourly_mean_service_time.mean() + 60)
print('\nHourly Service Rate for Wednesdays:',\
    round(mean_hourly_service_rate,3))
```

```
# use 15 calls per hour as the rate for one service operator
SERVICE_RATE = 15
# use a target for the probability of waiting in queue to be 0.50
PROBABILITY_GOAL = 0.50

# Erlang C queueing calculations with Python erlang_C function
# inputs c = number of servers
#        r = ratio of rate of arrivals and rate of service
# returns the propability of waiting in queue because all servers are busy
# use while-loop iteration to determine the number of servers needed
# we do this for each hour of the day knowing the hourly arrival rate
servers_needed = [0] * 24
for index_for_hour in range(24):
    if (hourly_arrival_rate[index_for_hour] > 0):
        erlang_probability = 1 # initialize on entering while-loop
        while (erlang_probability > PROBABILITY_GOAL):
            servers_needed[index_for_hour] = servers_needed[index_for_hour] + 1
            erlang_probability = \
                erlang_C(c = servers_needed[index_for_hour],\
                    r = hourly_arrival_rate[index_for_hour]/SERVICE_RATE)
print(servers_needed)  # check queueing theory result
# the result for servers.needed is obtained as
# 1  1  1  0  1  1  1  4  8  9 10  9  8 16 10 10  6  7  8  8  6  6  5  4
# we will assume the bank call center will be closed hours 00 through 05
# but use the other values as the bank's needed numbers of servers
for index_for_hour in range(6):
    servers_needed[index_for_hour] = 0
print('\nHourly Operator Requirements:\n',servers_needed)

# read in case data for the structure of call center worker shifts
bank_shifts_data_frame = pd.read_csv("data_anonymous_bank_shifts.csv")
# examine the structure of these data
print(bank_shifts_data_frame.head)

# constraint matrix as required for mathematical programming
constraint_matrix = np.array(bank_shifts_data_frame)[:,2:]
# we will create this type of object on the R side as well

# six-hour shift salaries in Israeli sheqels
# 1 ILS = 3.61 USD in June 2013
# these go into the objective function for integer programing
# with the objective of minimizing total costs
cost_vector = [252, 288, 180, 180, 180, 288, 288, 288]

# install lpsolove package and drivers for Python
# noting the operating system being used
# or use rpy2 access to lpSolve in R as shown here
# assign lists from Python to R using rpy2
r.assign('servers_needed_R', servers_needed)
r.assign('cost_vector_R', cost_vector)
r('bank.shifts.data.frame <- read.csv("data_anonymous_bank_shifts.csv")')
r('constraint_matrix_R <- as.matrix(bank.shifts.data.frame[,3:10])')
```

```
# check mathematical programming inputs on the R side
r('print(as.numeric(unlist(servers_needed_R)))')
r('print(as.numeric(unlist(cost_vector_R)))')
r('print(constraint_matrix_R)')

# solve the mathematical programming problem
r('library(lpSolve)')
r('call_center_schedule <- lp(const.mat=constraint_matrix_R,\
    const.rhs = as.numeric(unlist(servers_needed_R)),\
    const.dir = rep(">=", times = 8),\
    int.vec = 1:8,\
    objective = as.numeric(unlist(cost_vector_R)),\
    direction = "min")')

# prepare summary of the results for the call center problem
# working on the R side
r('ShiftID <- 1:8')
r('StartTime <- c(0,6,8,10,12,2,4,6)')
# c("Midnight","6 AM","8 AM","10 AM","Noon","2 PM","4 PM","6 PM")
r('ShiftDuration <- rep(6,times=8)')
r('HourlyShiftSalary <- c(42,48,30,30,30,48,48,48)')
r('HourlyShiftCost <- call_center_schedule$objective') # six x hourly shift salary
r('Solution <- call_center_schedule$solution')
r('ShiftCost <- call_center_schedule$solution * call_center_schedule$objective')

r('call_center_summary <- \
  data.frame(ShiftID,StartTime,ShiftDuration,HourlyShiftSalary,\
  HourlyShiftCost,Solution,ShiftCost)')

r('cat("\n\n","Call Center Summary","\n\n")')
r('print(call_center_summary)')
r('print(call_center_schedule)')

# alternatively... bring the solution from R to Python
# and print the minimum-cost solution on the Python side
call_center_schedule = r('call_center_schedule')
print(call_center_schedule)

# Suggestion for the student:
# Attack the problem using discrete event simulation,
# perhaps drawing on the SimPy package.
# Try running a sensitivity test, varying the workforce requirements
# and noting the effect upon the optimal assignment of workers to shifts.
# This can be done in a Python for-loop.
```

Exhibit 6.2. *Call Center Scheduling (R)*

```
# Workforce Scheduling for Anonymous Bank Call Center (R)
library(lubridate)  # date functions
library(grid)  # graphics utilities needed for split-plotting
library(ggplot2)  # graphics package with ribbon plot
library(queueing)  # queueing functions, including Erlang C
library(lpSolve)  # linear programming package
# ensure that two binary files are in the working directory
# these come from running R code from R_Utilities_Appendix
# source("R_utility_program_3.R") provides split-plotting utilities
load("mtpa_split_plotting_utilities.Rdata")
# source("R_utility_program_4.R") provides wait-time ribbon plots
load("mtpa_wait_time_ribbon_utility.Rdata")
put.title.on.plots <- TRUE  # put title on wait-time ribbon plots
# The call center data from "Anonymous Bank" in Israel were provided
# by Avi Mandelbaum, with the help of Ilan Guedj.
# data source: http://ie.technion.ac.il/serveng/callcenterdata/index.html
# variable names and definitions from documentation
# VRU  Voice Response Unit automated service
# vru.line  6 digits Each entering phone-call is first routed through a VRU:
#           There are 6 VRUs labeled AA01 to AA06. Each VRU has several lines
#           labeled 1-16. There are a total of 65 lines. Each call is assigned
#           a VRU number and a line number.
# call.id  unique call identifier
# customer.id  unique identifier for existing customer, zero for non-customer
# priority  0 or 1 for unidentified or regular customers
#           2 for priority customers who receive advanced position in queue
# type  type of service
#       PS  regular activity (coded 'PS' for 'Peilut Shotefet')
#       PE  regular activity in English (coded 'PE' for 'Peilut English')
#       IN  internet consulting (coded 'IN' for 'Internet')
#       NE  stock exchange activity (coded 'NE' for 'Niarot Erech')
#       NW  potential customer getting information
#       TT  customers who left a message asking the bank to return their call
#           but, while the system returned their call, the calling-agent became
#           busy hence the customers were put on hold in the queue.
# date  year-month-day
# vru_entry  time that the phone-call enters the call-center or VRU
# vru_exit  time of exit from VRU directly to service or to queue
# vru_time  time in seconds spent in the VRU
#           (calculated by exit_time  entry_time)
# q_start  time of joining the queue (00:00:00 for customers who abandon VRU
#          or do not enter the queue)
# q_exit  time in seconds of exiting queue to receive service or abandonment
# q_time  time spent in queue (calculated by q_exit  q_start)
# outcome  AGENT = service
#          HANG = hang up
#          PHANTOM = a virtual call to be ignored
# ser_start  time of beginning of service by agent
# ser_exit  time of end of service by agent
# ser_time  service duration in seconds (calculated by ser_exit  ser_start)
# server  name of agent, NO_SERVER if no service provided
```

```
# focus upon February 1999
call.center.input.data <- read.table("data_anonymous_bank_february.txt",
  header = TRUE, colClasses = c("character","integer","numeric",
  "integer","character","character","character","character","integer",
  "character","character","integer","factor","character","character",
  "integer","character"))

# check data frame object and variable values
print(summary(call.center.input.data))

# delete PHANTOM calls
call.center.data <- subset(call.center.input.data, subset = (outcome != "PHANTOM"))

# negative VRU times make no sense... drop these rows from data frame
call.center.data <- subset(call.center.data, subset = (vru_time >= 0))

# calculate wait time as sum of vru_time and q_time
call.center.data$wait_time <-
  call.center.data$vru_time + call.center.data$q_time

# define four-digit year so year is not read as 2099
# convert date string to date variable
call.center.data$date <- paste("19", call.center.data$date, sep ="")
call.center.data$date <- ymd(call.center.data$date)

# identify day of the week 1 = Sunday ... 7 = Saturday
call.center.data$day_of_week <- wday(call.center.data$date)
call.center.data$day_of_week <- factor(call.center.data$day_of_week,
  levels = c(1:7), labels = c("Sunday","Monday","Tuesday",
  "Wednesday","Thursday","Friday","Saturday"))

# examine frequency of calls by day of week
print(table(call.center.data$day_of_week))

# identify the hour of entry into the system
time.list <- strsplit(call.center.data$vru_entry,":")
call.hour <- numeric(nrow(call.center.data))
for (index.for.call in 1:nrow(call.center.data))
  call.hour[index.for.call] <- as.numeric(time.list[[index.for.call]][1])
call.center.data$call_hour <- call.hour
# check frequency of calls in February by hour and day of week
print(with(call.center.data, table(day_of_week, call_hour)))

# select first week of February 1999 for data visualization and analysis
# that week began on Monday, February 1 and ended on Sunday, February 7
selected.week <- subset(call.center.data, subset = (date < ymd("19990208")))

# check frequency of calls in week by hour and day of week
print(with(selected.week, table(day_of_week, call_hour)))
# loop for day of week ignoring Saturdays in Isreal
day.of.week.list <- c("Monday","Tuesday",
  "Wednesday","Thursday","Friday","Sunday")
```

```
# wait-time ribbon plots for the six selected days
# call upon utility function wait.time.ribbon
# the utility makes use of grid split-plotting
# place ribbon plot and text table/plot on each file
# each plot goes to its own external pdf file
for(index.day in seq(along=day.of.week.list)) {
  this.day.of.week <- day.of.week.list[index.day]
  pdf(file = paste("fig_operations_management_ribbon_",
  tolower(this.day.of.week),".pdf",sep=""), width = 11, height = 8.5)
  if(put.title.on.plots) {
    ribbon.plot.title <- paste(this.day.of.week,"Call Center Operations")
    }
    else {
    ribbon.plot.title <- ""
    }
  selected.day <- subset(selected.week,
    subset = (day_of_week == this.day.of.week),
    select = c("call_hour","wait_time","ser_time","server"))
  colnames(selected.day) <- c("hour","wait","service","server")
  wait.time.ribbon(wait.service.data = selected.day,
    title = ribbon.plot.title,
    use.text.tagging = TRUE, wait.time.goal = 30, wait.time.max = 90,
    plotting.min = 0, plotting.max = 250)
  dev.off()
  }

# select Wednesdays in February for the queueing model
wednesdays <- subset(call.center.data, subset = (day_of_week == "Wednesday"))

# compute arrival rate of calls as calls for hour
# we do not use table() here because some hours could have zero calls

calls.for.hour <- numeric(24)

for(index.for.hour in 1:24) {
# 24-hour clock has first hour coded as zero in input data file
  coded.index.for.hour <- index.for.hour - 1
  this.hour.calls <-
    subset(wednesdays, subset = (call_hour == coded.index.for.hour))
  if(nrow(this.hour.calls) > 0)
    calls.for.hour[index.for.hour] <- nrow(this.hour.calls)
  }

# compute arrival rate as average number of calls into VRU per hour
hourly.arrival.rate <- calls.for.hour/4  # four Wednesdays in February

# service times can vary hour-by-hour due to differences
# in service requests and individuals calling hour-by-hour
# begin by selecting calls that receive service

wednesdays.served <- subset(wednesdays, subset = (server != "NO_SERVER"))
```

```
hourly.mean.service.time <- numeric(24)
served.for.hour <- numeric(24)
for(index.for.hour in 1:24) {
# 24-hour clock has first hour coded as zero in input data file
  coded.index.for.hour <- index.for.hour - 1
  this.hour.calls <-
    subset(wednesdays.served, subset = (call_hour == coded.index.for.hour))
  if(nrow(this.hour.calls) > 0) {
    served.for.hour[index.for.hour] <- nrow(this.hour.calls)
    hourly.mean.service.time[index.for.hour] <- mean(this.hour.calls$ser_time)
    }
  }
# hourly service rate given the current numbers of service operators
hourly.served.rate <- served.for.hour/4  # four Wednesdays in February
# build data frame for plotting arrival and service rates
hour <- 1:24  # hour for horizontal axix of line chart
type <- rep("Arrived", length = 24)
value <- hourly.arrival.rate
arrival.data.frame <- data.frame(hour, value, type)
type <- rep("Served", length = 24)
value <- hourly.served.rate
service.data.frame <- data.frame(hour, value, type)
arrival.service.data.frame <- rbind(arrival.data.frame, service.data.frame)

pdf(file = "fig_operations_management_wednesdays_arrived_served.pdf",
  width = 11, height = 8.5)
plotting.object <- ggplot(data = arrival.service.data.frame,
  aes(x = hour, y = value, fill = type)) +
  geom_line() +
  geom_point(size = 4, shape = 21) +
  scale_x_continuous(breaks = c(1,3,5,7,9,11,13,15,17,19,21,23,25),
    labels =
      c("00","02","04","06","08","10","12","14","16","18","20","22","24")) +
  theme(axis.text.x = element_text(angle = 30, hjust = 1, vjust = 1)) +
  labs(x = "Hour of Day (24-Hour Clock)", y = "Average Calls per Hour") +
  scale_fill_manual(values = c("yellow","dark green"),
    guide = guide_legend(title = NULL))  +
  theme(legend.position = c(1,1), legend.justification = c(1,1)) +
  theme(legend.text = element_text(size=15)) +
  coord_fixed(ratio = 1/10)
print(plotting.object)
dev.off()
# examine service times per service operator
# for hours with no service time information use the mean as value
hourly.mean.service.time <-
  ifelse((hourly.mean.service.time == 0),
    mean(wednesdays.served$ser_time),
    hourly.mean.service.time)
# compute service rate noting that there are 3600 seconds in an hour
# adding 60 seconds to each mean service time for time between calls
# this 60 seconds is the wrap up time or time a service agent remains
# unavailable to answer a new call after a call has been completed
hourly.service.rate <- 3600/(hourly.mean.service.time + 60)
```

```
# we observe that mean service times do not vary that much hour-by-hour
# so we use the mean hourly service rate in queueing calculations
# mean(hourly.service.rate) is 14.86443
# so we use 15 calls per hour as the rate for one service operator
SERVICE.RATE <- 15
# C_erlang function from the queueing package
# inputs c = number of servers
#        r = ratio of rate of arrivals and rate of service
# returns the propability of waiting in queue because all servers are busy
# let us set a target for the probability of waiting in queue to be 0.50
# using while-loop iteration we determine the number of servers needed
# we do this for each hour of the day knowing the hourly arrival rate
PROBABILITY.GOAL <- 0.50
servers.needed <- integer(24)  # initialize to zero
for(index.for.hour in 1:24) {
  if (hourly.arrival.rate[index.for.hour] > 0) {
    erlang.probability <- 1.00  # intialization prior to entering while-loop
    while (erlang.probability > PROBABILITY.GOAL) {
      servers.needed[index.for.hour] <- servers.needed[index.for.hour] + 1
      erlang.probability <- C_erlang(c = servers.needed[index.for.hour],
          r = hourly.arrival.rate[index.for.hour]/SERVICE.RATE)
      }  # end while-loop for defining servers needed given probability goal
    }  # end if-block for hours with calls
  }  # end for-loop for the hour
# the result for servers.needed is obtained as
# 1  1  1  0  1  1  1  4  8  9 10  9  8 16 10 10  6  7  8  8  6  6  5  4
# we will assume the bank call center will be closed hours 00 through 05
# but use the other values as the bank's needed numbers of servers
servers.needed[1:6] <- 0
cat("\n","----- Hourly Operator Requirements -----","\n")
print(servers.needed)
# read in case data for the structure of call center worker shifts
bank.shifts.data.frame <- read.csv("data_anonymous_bank_shifts.csv")

# examine the structure of the case data frame
print(str(bank.shifts.data.frame))

constraint.matrix <- as.matrix(bank.shifts.data.frame[,3:10])
cat("\n","----- Call Center Shift Constraint Matrix -----","\n")
print(constraint.matrix)

# six-hour shift salaries in Israeli sheqels
# 1 ILS = 3.61 USD in June 2013
# these go into the objective function for integer programing
# with the objective of minimizing total costs
cost.vector <- c(252,288,180,180,180,288,288,288)
call.center.schedule <- lp(const.mat=constraint.matrix,
const.rhs = servers.needed,
const.dir = rep(">=",times=8),
int.vec = 1:8,
objective = cost.vector,
direction = "min")
```

```
# prepare summary of the results for the call center problem
ShiftID <- 1:8
StartTime <- c(0,6,8,10,12,2,4,6)
# c("Midnight","6 AM","8 AM","10 AM","Noon","2 PM","4 PM","6 PM")
ShiftDuration <- rep(6,times=8)
HourlyShiftSalary <- c(42,48,30,30,30,48,48,48)
HourlyShiftCost <- call.center.schedule$objective # six x hourly shift salary
Solution <- call.center.schedule$solution
ShiftCost <- call.center.schedule$solution * call.center.schedule$objective

call.center.summary <-
  data.frame(ShiftID,StartTime,ShiftDuration,HourlyShiftSalary,
  HourlyShiftCost,Solution,ShiftCost)
cat("\n\n","Call Center Summary","\n\n")
print(call.center.summary)
# the solution is obtained by print(call.center.schedule)
# or by summing across the hourly solution times the cost objective
 print(call.center.schedule)
cat("\n\n","Call Center Summary Minimum Cost Solution:",sum(ShiftCost),"\n\n")
# build data frame for plotting the solution compared with need
hour <- 1:24  # hour for horizontal axix of line chart
type <- rep("Hourly Need", length = 24)
value <- servers.needed
needs.data.frame <- data.frame(hour, value, type)
type <- rep("Optimal Solution", length = 24)
value <- schedule.fit.to.need <-
  constraint.matrix %*% call.center.schedule$solution
solution.data.frame <- data.frame(hour, value, type)
plotting.data.frame <- rbind(needs.data.frame, solution.data.frame)
# plot the solution... solution match to the workforce need
pdf(file = "fig_operations_management_solution.pdf", width = 11, height = 8.5)
plotting.object <- ggplot(data = plotting.data.frame,
  aes(x = hour, y = value, fill = type)) +
  geom_line() +
  geom_point(size = 4, shape = 21) +
  scale_x_continuous(breaks = c(1,3,5,7,9,11,13,15,17,19,21,23,25),
    labels =
      c("00","02","04","06","08","10","12","14","16","18","20","22","24")) +
  theme(axis.text.x = element_text(angle = 30, hjust = 1, vjust = 1)) +
  labs(x = "Hour of Day (24-Hour Clock)", y = "Number of Service Operators") +
  scale_fill_manual(values = c("white","blue"),
    guide = guide_legend(title = NULL)) +
  theme(legend.position = c(1,1), legend.justification = c(1,1)) +
  theme(legend.text = element_text(size=15)) +
  coord_fixed(ratio = 2/2.25)
print(plotting.object)
dev.off()
```

Exhibit D.6. *Split-plotting Utilities (R)*

```
# Split-Plotting Utilities with grid Graphics (R)

library(grid)  # grid graphics foundation of split-plotting utilities

# functions used with ggplot2 graphics to split the plotting region
# to set margins and to plot more than one ggplot object on one page/screen

vplayout <- function(x, y)
viewport(layout.pos.row=x, layout.pos.col=y)


# grid graphics utility plots one plot with margins
ggplot.print.with.margins <- function(ggplot.object.name,left.margin.pct=10,
  right.margin.pct=10,top.margin.pct=10,bottom.margin.pct=10)
{ # begin function for printing ggplot objects with margins
  # margins expressed as percentages of total... use integers
 grid.newpage()
pushViewport(viewport(layout=grid.layout(100,100)))
print(ggplot.object.name,
  vp=vplayout((0 + top.margin.pct):(100 - bottom.margin.pct),
  (0 + left.margin.pct):(100 - right.margin.pct)))
} # end function for printing ggplot objects with margins



# grid graphics utility plots two ggplot plotting objects in one column
special.top.bottom.ggplot.print.with.margins <-
  function(ggplot.object.name,ggplot.text.tagging.object.name,
  left.margin.pct=5,right.margin.pct=5,top.margin.pct=5,
  bottom.margin.pct=5,plot.pct=80,text.tagging.pct=10) {
# begin function for printing ggplot objects with margins
# and text tagging at bottom of plot
# margins expressed as percentages of total... use integers
  if((top.margin.pct + bottom.margin.pct + plot.pct + text.tagging.pct) != 100)
    stop(paste("function special.top.bottom.ggplot.print.with.margins()",
    "execution terminated:\n   top.margin.pct + bottom.margin.pct + ",
    "plot.pct + text.tagging.pct not equal to 100 percent",sep=""))
  grid.newpage()
  pushViewport(viewport(layout=grid.layout(100,100)))
  print(ggplot.object.name,
  vp=vplayout((0 + top.margin.pct):
    (100 - (bottom.margin.pct + text.tagging.pct)),
  (0 + left.margin.pct):(100 - right.margin.pct)))

  print(ggplot.text.tagging.object.name,
    vp=vplayout((0 + (top.margin.pct + plot.pct)):(100 - bottom.margin.pct),
    (0 + left.margin.pct):(100 - right.margin.pct)))
} # end function for printing ggplot objects with margins and text tagging
```

```
# grid graphics utility plots three ggplot plotting objects in one column
three.part.ggplot.print.with.margins <- function(ggfirstplot.object.name,
ggsecondplot.object.name,
ggthirdplot.object.name,
left.margin.pct=5,right.margin.pct=5,
top.margin.pct=10,bottom.margin.pct=10,
first.plot.pct=25,second.plot.pct=25,
third.plot.pct=30) {
# function for printing ggplot objects with margins and top and bottom plots
# margins expressed as percentages of total... use integers
if((top.margin.pct + bottom.margin.pct + first.plot.pct +
  second.plot.pct  + third.plot.pct) != 100)
    stop(paste("function special.top.bottom.ggplot.print.with.margins()",
         "execution terminated:\n   top.margin.pct + bottom.margin.pct",
         "+ first.plot.pct + second.plot.pct  + third.plot.pct not equal",
         "to 100 percent",sep=""))
grid.newpage()
pushViewport(viewport(layout=grid.layout(100,100)))

print(ggfirstplot.object.name, vp=vplayout((0 + top.margin.pct):
  (100 - (second.plot.pct  + third.plot.pct + bottom.margin.pct)),
  (0 + left.margin.pct):(100 - right.margin.pct)))

print(ggsecondplot.object.name,
  vp=vplayout((0 + top.margin.pct + first.plot.pct):
  (100 - (third.plot.pct + bottom.margin.pct)),
  (0 + left.margin.pct):(100 - right.margin.pct)))

print(ggthirdplot.object.name,
  vp=vplayout((0 + top.margin.pct + first.plot.pct + second.plot.pct):
  (100 - (bottom.margin.pct)),(0 + left.margin.pct):
  (100 - right.margin.pct)))
}

# grid graphics utility plots two ggplot plotting objects in one row
# primary plot graph at left... legend at right
special.left.right.ggplot.print.with.margins <-
  function(ggplot.object.name, ggplot.text.legend.object.name,
  left.margin.pct=5, right.margin.pct=5, top.margin.pct=5,
  bottom.margin.pct=5, plot.pct=85, text.legend.pct=5) {
# begin function for printing ggplot objects with margins
# and text legend at bottom of plot
# margins expressed as percentages of total... use integers
  if((left.margin.pct + right.margin.pct + plot.pct + text.legend.pct) != 100)
    stop(paste("function special.left.right.ggplot.print.with.margins()",
    "execution terminated:\n   left.margin.pct + right.margin.pct + ",
    "plot.pct + text.legend.pct not equal to 100 percent",sep=""))
  grid.newpage()
  pushViewport(viewport(layout=grid.layout(100,100)))
  print(ggplot.object.name,
  vp=vplayout((0 + top.margin.pct):(100 - (bottom.margin.pct)),
  (0 + left.margin.pct + text.legend.pct):(100 - right.margin.pct)))
```

```
  print(ggplot.text.legend.object.name,
    vp=vplayout((0 + (top.margin.pct)):(100 - bottom.margin.pct),
    (0 + left.margin.pct + plot.pct):(100 - right.margin.pct)))
} # end function for printing ggplot objects with margins and text legend

# save split-plotting utilities for future work
save(vplayout,
  ggplot.print.with.margins,
  special.top.bottom.ggplot.print.with.margins,
  three.part.ggplot.print.with.margins,
  special.left.right.ggplot.print.with.margins,
  file="mtpa_split_plotting_utilities.Rdata")
```

Exhibit D.7. *Wait-time Ribbon Plot (R)*

```
# Wait-Time Ribbon Plot (R)

wait.time.ribbon <- function(wait.service.data, title = "",
  wait.time.goal = 30, wait.time.max = 90,
  plotting.min = 0, plotting.max = 250,
  use.text.tagging = TRUE) {
#  requires ggplot2 package
#  data visualization for operations management
#  wait.service.data is input data frame with the named columns as follows:
#    hour: integer hour of the day on 24-hour clock
#    wait: integer call wait time in seconds
#    service:  integer call service time in seconds (NA for no service)
#    server:  character string for server name or code
#             assumes that there is a distict character string for no server
#             this string is coded as NO_SERVER
#  wait.time.goal:  desired maximum wait time (30 seconds default)
#                   represented as bottom of yellow region
#  wait.time.max: when wait time becomes intolerable (90 seconds default)
#                 represented as top of yellow region
# use.text.tagging default is TRUE for added text at bottom of plot
# set constants for ribbon plotting
MIN.SAMPLE <- 5  # min sample size for hourly calcuations
PERCENTILE.MIN <- 0.50  # used for bottom of acceptable wait time
PERCENTILE.MAX <- 0.90  # used for bottom of acceptable wait time
add_footnote_at_bottom_of_ribbon_plot <- TRUE
percentile.footnote <- paste("Bottom of ribbon = ",
  100*PERCENTILE.MIN, "th percentile of wait times",
  "    Top of ribbon = ", 100*PERCENTILE.MAX, "th percentile of wait times.",
  sep = "")

x.hour <- seq(from=0,to=23) # for horixontal axis scale

# code for ribbon region counts
calls.per.hour <- numeric(24)  # total calls initialized as zero
served.calls <- numeric(24)  # served calls initialized as zero
dropped.calls <- numeric(24)  # dropped/abandoned calls initialize as zero
ymin.percentile <- rep(NA,times=24)  # store for minimum percentile values
ymax.percentile <- rep(NA,times=24)  # store maximum percentile values

# compute number of calls per hour
# code more versatile than table command
# to accommodate hours with no calls
for(index.for.hour in 1:24) {
# begin for-loop for wait-time data call counts and percentile calculations
# 24-hour clock has first hour coded as zero in input data file
  coded.index.for.hour <- index.for.hour - 1
  temporary.vector <- na.omit(wait.service.data$hour)
  calls.per.hour[index.for.hour] <-
    sum(ifelse(temporary.vector==coded.index.for.hour,1,0))
    if(calls.per.hour[index.for.hour] >= MIN.SAMPLE) {
```

```
# begin if-block for computing ymin and ymax values and number of servers
# when there are at least MIN.SAMPLE calls in the hour
    this.hour.wait.service.data <-
      wait.service.data[(wait.service.data$hour == coded.index.for.hour),]

    ymin.percentile[index.for.hour] <-
      quantile(this.hour.wait.service.data$wait,
      probs=c(PERCENTILE.MIN),na.rm = TRUE,names=FALSE,type=8)

    ymax.percentile[index.for.hour] <-
      quantile(this.hour.wait.service.data$wait,
      probs=c(PERCENTILE.MAX),na.rm = TRUE,names=FALSE,type=8)
    } # end if-block for computing ymin and ymax values

# if insufficient data we set min and max to be wait.time.goal
  if(calls.per.hour[index.for.hour] < MIN.SAMPLE) {
    ymin.percentile[index.for.hour] <- wait.time.goal
    ymax.percentile[index.for.hour] <- wait.time.goal
    }
  } # end for-loop for wait-time data call counts and percentile calculations

# compute number.of.servers data and served and dropped calls
number.of.servers <- numeric(24)  # initialize to zero
for(index.for.hour in 1:24) {
# begin for-loop for obtaining server data for the ribbon plot
# 24-hour clock has first hour coded as zero in input data file
  coded.index.for.hour <- index.for.hour - 1
  temporary.vector <- na.omit(wait.service.data$hour)
  calls.per.hour[index.for.hour] <-
    sum(ifelse(temporary.vector==coded.index.for.hour,1,0))
  this.hour.wait.service.data <-
      wait.service.data[(wait.service.data$hour == coded.index.for.hour),]

  served.calls[index.for.hour] <-
    nrow(subset(this.hour.wait.service.data, subset=(server != "NO_SERVER")))
  dropped.calls[index.for.hour] <-
    nrow(subset(this.hour.wait.service.data, subset=(server == "NO_SERVER")))

  if (nrow(this.hour.wait.service.data) > 0) {
# count is based upon the number of unique server names less NO_SERVER
    servers <-
      na.omit((unique(this.hour.wait.service.data$server)))
    valid.servers <- setdiff(servers, "NO_SERVER")
    number.of.servers[index.for.hour] <- length(valid.servers)
    }
  } # end for-loop for obtaining server data for the ribbon plot

greenmin <- rep(plotting.min, length=24)
greenmax <- rep(wait.time.goal, length=24)

yellowmin <- rep(wait.time.goal, length=24)
yellowmax <- rep(wait.time.max, length=24)
```

```
redmin <- rep(wait.time.max, length=24)
redmax <- rep(plotting.max, length=24)

ymax.topwhite <- rep(plotting.max,length=24)
ymin.topwhite <- ymax.percentile

ymax.bottomwhite <- ymin.percentile
ymin.bottomwhite <- rep(plotting.min,length=24)

# define data frame for plotting wait and service information for this day

call.center.plotting.frame <-
  data.frame(x.hour, ymin.percentile, ymax.percentile,
    calls.per.hour, number.of.servers,
    greenmin,greenmax,
    yellowmin,yellowmax,
    redmin,redmax,
    ymin.bottomwhite,ymax.bottomwhite,
    ymin.topwhite,ymax.topwhite)

#cat("\n\n","------------- ",title," -------------","\n")
#print(call.center.plotting.frame)

ggobject <- ggplot() +
geom_ribbon(data=call.center.plotting.frame,
mapping=aes(x=x.hour, ymin=greenmin, ymax=greenmax),
stat="identity",colour="white",fill="darkgreen") +
geom_ribbon(data=call.center.plotting.frame,
mapping=aes(x=x.hour, ymin=yellowmin, ymax=yellowmax),
stat="identity",colour="white",fill="yellow") +
geom_ribbon(data=call.center.plotting.frame,
mapping=aes(x=x.hour, ymin=redmin, ymax=redmax),
stat="identity",colour="white",fill="red") +
geom_ribbon(data=call.center.plotting.frame,
mapping=aes(x=x.hour, ymin=ymin.topwhite, ymax=ymax.topwhite),
stat="identity",colour="white",fill="white") +
geom_ribbon(data=call.center.plotting.frame,
mapping=aes(x=x.hour, ymin=ymin.bottomwhite, ymax=ymax.bottomwhite),
stat="identity",colour="white",fill="white") +
geom_hline(data=call.center.plotting.frame,
mapping=aes(yintercept=yellowmin[1])) +
geom_hline(data=call.center.plotting.frame,
mapping=aes(yintercept=redmin[1])) +
labs(title = title) + theme_bw(base_size = 12) +
scale_y_continuous(limits = c(greenmin[1], redmax[1])) +
xlab("Hour of Day (24-Hour Clock)") +
ylab("Wait Time (Seconds)")

# plotting with all default margins no text at bottom
if(!use.text.tagging) ggplot.print.with.margins(ggobject)
```

```
# plotting with text tagging requires the creation of a ggplot text object
if (use.text.tagging) {
# define character data for the text taggging at bottom of plot
hour.title <- "Hour:"
hour.00 <- "00"
hour.01 <- "01"
hour.02 <- "02"
hour.03 <- "03"
hour.04 <- "04"
hour.05 <- "05"
hour.06 <- "06"
hour.07 <- "07"
hour.08 <- "08"
hour.09 <- "09"
hour.10 <- "10"
hour.11 <- "11"
hour.12 <- "12"
hour.13 <- "13"
hour.14 <- "14"
hour.15 <- "15"
hour.16 <- "16"
hour.17 <- "17"
hour.18 <- "18"
hour.19 <- "19"
hour.20 <- "20"
hour.21 <- "21"
hour.22 <- "22"
hour.23 <- "23"
calls.title <- "Calls:"
calls.00 <- as.character(calls.per.hour[1])
calls.01 <- as.character(calls.per.hour[2])
calls.02 <- as.character(calls.per.hour[3])
calls.03 <- as.character(calls.per.hour[4])
calls.04 <- as.character(calls.per.hour[5])
calls.05 <- as.character(calls.per.hour[6])
calls.06 <- as.character(calls.per.hour[7])
calls.07 <- as.character(calls.per.hour[8])
calls.08 <- as.character(calls.per.hour[9])
calls.09 <- as.character(calls.per.hour[10])
calls.10 <- as.character(calls.per.hour[11])
calls.11 <- as.character(calls.per.hour[12])
calls.12 <- as.character(calls.per.hour[13])
calls.13 <- as.character(calls.per.hour[14])
calls.14 <- as.character(calls.per.hour[15])
calls.15 <- as.character(calls.per.hour[16])
calls.16 <- as.character(calls.per.hour[17])
calls.17 <- as.character(calls.per.hour[18])
calls.18 <- as.character(calls.per.hour[19])
calls.19 <- as.character(calls.per.hour[20])
calls.20 <- as.character(calls.per.hour[21])
calls.21 <- as.character(calls.per.hour[22])
calls.22 <- as.character(calls.per.hour[23])
calls.23 <- as.character(calls.per.hour[24])
```

```
servers.title <- "Servers:"
servers.00 <- as.character(number.of.servers[1])
servers.01 <- as.character(number.of.servers[2])
servers.02 <- as.character(number.of.servers[3])
servers.03 <- as.character(number.of.servers[4])
servers.04 <- as.character(number.of.servers[5])
servers.05 <- as.character(number.of.servers[6])
servers.06 <- as.character(number.of.servers[7])
servers.07 <- as.character(number.of.servers[8])
servers.08 <- as.character(number.of.servers[9])
servers.09 <- as.character(number.of.servers[10])
servers.10 <- as.character(number.of.servers[11])
servers.11 <- as.character(number.of.servers[12])
servers.12 <- as.character(number.of.servers[13])
servers.13 <- as.character(number.of.servers[14])
servers.14 <- as.character(number.of.servers[15])
servers.15 <- as.character(number.of.servers[16])
servers.16 <- as.character(number.of.servers[17])
servers.17 <- as.character(number.of.servers[18])
servers.18 <- as.character(number.of.servers[19])
servers.19 <- as.character(number.of.servers[20])
servers.20 <- as.character(number.of.servers[21])
servers.21 <- as.character(number.of.servers[22])
servers.22 <- as.character(number.of.servers[23])
servers.23 <- as.character(number.of.servers[24])


served.title <- "Served:"
served.00 <- as.character(served.calls[1])
served.01 <- as.character(served.calls[2])
served.02 <- as.character(served.calls[3])
served.03 <- as.character(served.calls[4])
served.04 <- as.character(served.calls[5])
served.05 <- as.character(served.calls[6])
served.06 <- as.character(served.calls[7])
served.07 <- as.character(served.calls[8])
served.08 <- as.character(served.calls[9])
served.09 <- as.character(served.calls[10])
served.10 <- as.character(served.calls[11])
served.11 <- as.character(served.calls[12])
served.12 <- as.character(served.calls[13])
served.13 <- as.character(served.calls[14])
served.14 <- as.character(served.calls[15])
served.15 <- as.character(served.calls[16])
served.16 <- as.character(served.calls[17])
served.17 <- as.character(served.calls[18])
served.18 <- as.character(served.calls[19])
served.19 <- as.character(served.calls[20])
served.20 <- as.character(served.calls[21])
served.21 <- as.character(served.calls[22])
served.22 <- as.character(served.calls[23])
served.23 <- as.character(served.calls[24])
```

```
dropped.title <- "Dropped:"
dropped.00 <- as.character(dropped.calls[1])
dropped.01 <- as.character(dropped.calls[2])
dropped.02 <- as.character(dropped.calls[3])
dropped.03 <- as.character(dropped.calls[4])
dropped.04 <- as.character(dropped.calls[5])
dropped.05 <- as.character(dropped.calls[6])
dropped.06 <- as.character(dropped.calls[7])
dropped.07 <- as.character(dropped.calls[8])
dropped.08 <- as.character(dropped.calls[9])
dropped.09 <- as.character(dropped.calls[10])
dropped.10 <- as.character(dropped.calls[11])
dropped.11 <- as.character(dropped.calls[12])
dropped.12 <- as.character(dropped.calls[13])
dropped.13 <- as.character(dropped.calls[14])
dropped.14 <- as.character(dropped.calls[15])
dropped.15 <- as.character(dropped.calls[16])
dropped.16 <- as.character(dropped.calls[17])
dropped.17 <- as.character(dropped.calls[18])
dropped.18 <- as.character(dropped.calls[19])
dropped.19 <- as.character(dropped.calls[20])
dropped.20 <- as.character(dropped.calls[21])
dropped.21 <- as.character(dropped.calls[22])
dropped.22 <- as.character(dropped.calls[23])
dropped.23 <- as.character(dropped.calls[24])

# set up spacing and positioning for the table
y.current.level <- 1.0  # initialze position
y.large.space <- 0.175
y.medium.space <- 0.125
y.small.space <- 0.075

table.left.margin <- 0.1  # needed for row labels at left
horizontal.offset <- (1-table.left.margin)/24  # spacing in the text table

y.current.level <- y.current.level - y.medium.space

ggtextobject <- ggplot(data=data.frame(x = 0.5,y = y.current.level),
  aes(x=x,y=y,xmin=0,xmax=1,ymin=0,ymax=1),
  stat="identity", position="identity") + labs(x=NULL,y=NULL) +
geom_text(x = 0.025,y = y.current.level,label = hour.title,
aes(size=10.55),colour="black")  +
geom_text(x = (00*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.00, aes(size=10.55),colour="black") +
geom_text(x = (01*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.01, aes(size=10.55),colour="black") +
geom_text(x = (02*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.02, aes(size=10.55),colour="black") +
geom_text(x = (03*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.03, aes(size=10.55),colour="black") +
geom_text(x = (04*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.04, aes(size=10.55),colour="black") +
```

```
geom_text(x = (05*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.05, aes(size=10.55),colour="black") +
geom_text(x = (06*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.06, aes(size=10.55),colour="black") +
geom_text(x = (07*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.07, aes(size=10.55),colour="black") +
geom_text(x = (08*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.08, aes(size=10.55),colour="black") +
geom_text(x = (09*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.09, aes(size=10.55),colour="black") +
geom_text(x = (10*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.10, aes(size=10.55),colour="black") +
geom_text(x = (11*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.11, aes(size=10.55),colour="black") +
geom_text(x = (12*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.12, aes(size=10.55),colour="black") +
geom_text(x = (13*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.13, aes(size=10.55),colour="black") +
geom_text(x = (14*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.14, aes(size=10.55),colour="black") +
geom_text(x = (15*horizontal.offset + table.left.margin),
y= y.current.level,label = hour.15, aes(size=10.55),colour="black") +
geom_text(x = (16*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.16, aes(size=10.55),colour="black") +
geom_text(x = (17*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.17, aes(size=10.55),colour="black") +
geom_text(x = (18*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.18, aes(size=10.55),colour="black") +
geom_text(x = (19*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.19, aes(size=10.55),colour="black") +
geom_text(x = (20*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.20, aes(size=10.55),colour="black") +
geom_text(x = (21*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.21, aes(size=10.55),colour="black") +
geom_text(x = (22*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.22, aes(size=10.55),colour="black") +
geom_text(x = (23*horizontal.offset + table.left.margin),
y = y.current.level,label = hour.23, aes(size=10.55),colour="black")

y.current.level <- y.current.level - y.medium.space

ggtextobject <- ggtextobject + geom_text(x = 0.025,
y = y.current.level, label = servers.title,aes(size=10.55),colour="black") +
geom_text(x = (00*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.00,aes(size=10.55),colour="black") +
geom_text(x = (01*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.01,aes(size=10.55),colour="black") +
geom_text(x = (02*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.02,aes(size=10.55),colour="black") +
geom_text(x = (03*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.03,aes(size=10.55),colour="black") +
geom_text(x = (04*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.04,aes(size=10.55),colour="black") +
```

```
geom_text(x = (05*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.05,aes(size=10.55),colour="black") +
geom_text(x = (06*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.06,aes(size=10.55),colour="black") +
geom_text(x = (07*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.07,aes(size=10.55),colour="black") +
geom_text(x = (08*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.08,aes(size=10.55),colour="black") +
geom_text(x = (09*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.09,aes(size=10.55),colour="black") +
geom_text(x = (10*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.10,aes(size=10.55),colour="black") +
geom_text(x = (11*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.11,aes(size=10.55),colour="black") +
geom_text(x = (12*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.12,aes(size=10.55),colour="black") +
geom_text(x = (13*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.13,aes(size=10.55),colour="black") +
geom_text(x = (14*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.14,aes(size=10.55),colour="black") +
geom_text(x = (15*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.15,aes(size=10.55),colour="black") +
geom_text(x = (16*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.16,aes(size=10.55),colour="black") +
geom_text(x = (17*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.17,aes(size=10.55),colour="black") +
geom_text(x = (18*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.18,aes(size=10.55),colour="black") +
geom_text(x = (19*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.19,aes(size=10.55),colour="black") +
geom_text(x = (20*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.20,aes(size=10.55),colour="black") +
geom_text(x = (21*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.21,aes(size=10.55),colour="black") +
geom_text(x = (22*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.22,aes(size=10.55),colour="black") +
geom_text(x = (23*horizontal.offset + table.left.margin),
y = y.current.level, label = servers.23,aes(size=10.55),colour="black")

# store line position for bottom of text segment of the visualization

y.level.divider.line <- y.current.level - y.medium.space
# temporary data frame needed to input to geom_hline later
middle.line.data <- data.frame(y.level.divider.line)

y.current.level <- y.level.divider.line - y.medium.space

ggtextobject <- ggtextobject +
geom_text(x = 0.025,y = y.current.level,
label = calls.title, aes(size=10.55),colour="black") +
geom_text(x = (00*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.00, aes(size=10.55),colour="black") +
```

```
geom_text(x = (01*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.01, aes(size=10.55),colour="black") +
geom_text(x = (02*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.02, aes(size=10.55),colour="black") +
geom_text(x = (03*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.03, aes(size=10.55),colour="black") +
geom_text(x = (04*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.04, aes(size=10.55),colour="black") +
geom_text(x = (05*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.05, aes(size=10.55),colour="black") +
geom_text(x = (06*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.06, aes(size=10.55),colour="black") +
geom_text(x = (07*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.07, aes(size=10.55),colour="black") +
geom_text(x = (08*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.08, aes(size=10.55),colour="black") +
geom_text(x = (09*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.09, aes(size=10.55),colour="black") +
geom_text(x = (10*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.10, aes(size=10.55),colour="black") +
geom_text(x = (11*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.11, aes(size=10.55),colour="black") +
geom_text(x = (12*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.12, aes(size=10.55),colour="black") +
geom_text(x = (13*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.13, aes(size=10.55),colour="black") +
geom_text(x = (14*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.14, aes(size=10.55),colour="black") +
geom_text(x = (15*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.15, aes(size=10.55),colour="black") +
geom_text(x = (16*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.16, aes(size=10.55),colour="black") +
geom_text(x = (17*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.17, aes(size=10.55),colour="black") +
geom_text(x = (18*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.18, aes(size=10.55),colour="black") +
geom_text(x = (19*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.19, aes(size=10.55),colour="black") +
geom_text(x = (20*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.20, aes(size=10.55),colour="black") +
geom_text(x = (21*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.21, aes(size=10.55),colour="black") +
geom_text(x = (22*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.22, aes(size=10.55),colour="black") +
geom_text(x = (23*horizontal.offset + table.left.margin),
y = y.current.level, label = calls.23, aes(size=10.55),colour="black")

y.current.level <- y.current.level - y.medium.space

ggtextobject <- ggtextobject +
geom_text(x = 0.025,y = y.current.level,
label = served.title, aes(size=10.55),colour="black") +
```

```
geom_text(x = (00*horizontal.offset + table.left.margin),
y = y.current.level, label = served.00, aes(size=10.55),colour="black") +
geom_text(x = (01*horizontal.offset + table.left.margin),
y = y.current.level, label = served.01, aes(size=10.55),colour="black") +
geom_text(x = (02*horizontal.offset + table.left.margin),
y = y.current.level, label = served.02, aes(size=10.55),colour="black") +
geom_text(x = (03*horizontal.offset + table.left.margin),
y = y.current.level, label = served.03, aes(size=10.55),colour="black") +
geom_text(x = (04*horizontal.offset + table.left.margin),
y = y.current.level, label = served.04, aes(size=10.55),colour="black") +
geom_text(x = (05*horizontal.offset + table.left.margin),
y = y.current.level, label = served.05, aes(size=10.55),colour="black") +
geom_text(x = (06*horizontal.offset + table.left.margin),
y = y.current.level, label = served.06, aes(size=10.55),colour="black") +
geom_text(x = (07*horizontal.offset + table.left.margin),
y = y.current.level, label = served.07, aes(size=10.55),colour="black") +
geom_text(x = (08*horizontal.offset + table.left.margin),
y = y.current.level, label = served.08, aes(size=10.55),colour="black") +
geom_text(x = (09*horizontal.offset + table.left.margin),
y = y.current.level, label = served.09, aes(size=10.55),colour="black") +
geom_text(x = (10*horizontal.offset + table.left.margin),
y = y.current.level, label = served.10, aes(size=10.55),colour="black") +
geom_text(x = (11*horizontal.offset + table.left.margin),
y = y.current.level, label = served.11, aes(size=10.55),colour="black") +
geom_text(x = (12*horizontal.offset + table.left.margin),
y = y.current.level, label = served.12, aes(size=10.55),colour="black") +
geom_text(x = (13*horizontal.offset + table.left.margin),
y = y.current.level, label = served.13, aes(size=10.55),colour="black") +
geom_text(x = (14*horizontal.offset + table.left.margin),
y = y.current.level, label = served.14, aes(size=10.55),colour="black") +
geom_text(x = (15*horizontal.offset + table.left.margin),
y = y.current.level, label = served.15, aes(size=10.55),colour="black") +
geom_text(x = (16*horizontal.offset + table.left.margin),
y = y.current.level, label = served.16, aes(size=10.55),colour="black") +
geom_text(x = (17*horizontal.offset + table.left.margin),
y = y.current.level, label = served.17, aes(size=10.55),colour="black") +
geom_text(x = (18*horizontal.offset + table.left.margin),
y = y.current.level, label = served.18, aes(size=10.55),colour="black") +
geom_text(x = (19*horizontal.offset + table.left.margin),
y = y.current.level, label = served.19, aes(size=10.55),colour="black") +
geom_text(x = (20*horizontal.offset + table.left.margin),
y = y.current.level, label = served.20, aes(size=10.55),colour="black") +
geom_text(x = (21*horizontal.offset + table.left.margin),
y = y.current.level, label = served.21, aes(size=10.55),colour="black") +
geom_text(x = (22*horizontal.offset + table.left.margin),
y = y.current.level, label = served.22, aes(size=10.55),colour="black") +
geom_text(x = (23*horizontal.offset + table.left.margin),
y = y.current.level, label = served.23, aes(size=10.55),colour="black")

y.current.level <- y.current.level - y.medium.space

ggtextobject <- ggtextobject +
```

```
geom_text(x = 0.025,y = y.current.level,
label = dropped.title, aes(size=10.55),colour="black") +
geom_text(x = (00*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.00, aes(size=10.55),colour="black") +
geom_text(x = (01*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.01, aes(size=10.55),colour="black") +
geom_text(x = (02*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.02, aes(size=10.55),colour="black") +
geom_text(x = (03*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.03, aes(size=10.55),colour="black") +
geom_text(x = (04*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.04, aes(size=10.55),colour="black") +
geom_text(x = (05*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.05, aes(size=10.55),colour="black") +
geom_text(x = (06*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.06, aes(size=10.55),colour="black") +
geom_text(x = (07*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.07, aes(size=10.55),colour="black") +
geom_text(x = (08*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.08, aes(size=10.55),colour="black") +
geom_text(x = (09*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.09, aes(size=10.55),colour="black") +
geom_text(x = (10*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.10, aes(size=10.55),colour="black") +
geom_text(x = (11*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.11, aes(size=10.55),colour="black") +
geom_text(x = (12*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.12, aes(size=10.55),colour="black") +
geom_text(x = (13*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.13, aes(size=10.55),colour="black") +
geom_text(x = (14*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.14, aes(size=10.55),colour="black") +
geom_text(x = (15*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.15, aes(size=10.55),colour="black") +
geom_text(x = (16*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.16, aes(size=10.55),colour="black") +
geom_text(x = (17*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.17, aes(size=10.55),colour="black") +
geom_text(x = (18*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.18, aes(size=10.55),colour="black") +
geom_text(x = (19*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.19, aes(size=10.55),colour="black") +
geom_text(x = (20*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.20, aes(size=10.55),colour="black") +
geom_text(x = (21*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.21, aes(size=10.55),colour="black") +
geom_text(x = (22*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.22, aes(size=10.55),colour="black") +
geom_text(x = (23*horizontal.offset + table.left.margin),
y = y.current.level, label = dropped.23, aes(size=10.55),colour="black")

y.level.divider.line <- y.current.level - y.medium.space
```

```
# temporary data frame needed to input to geom_hline later
bottom.line.data <- data.frame(y.level.divider.line)

y.current.level <- y.level.divider.line - y.medium.space

# add footnote centered at bottom of plot if requested
if (add_footnote_at_bottom_of_ribbon_plot)
  ggtextobject <- ggtextobject +
  geom_text(x = 0.5,y = y.current.level,
  label = percentile.footnote, aes(size=10.55), colour="black")

# finish up the plot with background definition and divider lines
ggtextobject <- ggtextobject + geom_hline(aes(yintercept=1)) +
geom_hline(data=bottom.line.data,
  mapping = aes(yintercept = y.level.divider.line)) +
geom_hline(data=middle.line.data,
  mapping = aes(yintercept = y.level.divider.line)) +
theme(legend.position = "none")  +
theme(panel.grid.minor = element_blank()) +
theme(panel.grid.major = element_blank())  +
theme(panel.background = element_blank()) +
theme(axis.ticks = element_blank()) +
scale_y_continuous(breaks=c(0,1),label=c("","")) +
scale_x_continuous(breaks=c(0,1),label=c("",""))

# user-defined function plots with text annotation/tagging at the bottom
special.top.bottom.ggplot.print.with.margins(ggobject,ggtextobject,
plot.pct=55,text.tagging.pct=35)
}
} # end of wait-time ribbon function

# save wait-time ribbon utility for future work
save(wait.time.ribbon,
  file="mtpa_wait_time_ribbon_utility.Rdata")
```