Session 1: Exploratory Data Analysis

Natalie Nelson, PhD, Biological & Agricultural Engineering, NCSU

01/19/2018

1 Data

The dataset analyzed in this exercise includes Air Quality Index (AQI) values measured daily in 2017 across over 800 counties in the U.S. All 50 states are represented within the dataset. Note that measurements were not taken on each day in 2017. The original data was procured by the U.S. Environmental Protection Agency (EPA), and can be accessed from the EPA's Air Data webpage. Some observations have been removed in the exercise dataset in order to avoid coding problems.

Information on the AQI from the EPA (quoted from *EPA-456/F-14-002*)

What is the AQI?

The AQI is an index for reporting daily air quality. It tells you how clean or unhealthy your air is, and what associated health effects might be a concern. The AQI focuses on health effects you may experience within a few hours or days after breathing unhealthy air. The AQI is calculated for four major air pollutants regulated by the Clean Air Act: groundlevel ozone, particle pollution, carbon monoxide, and sulfur dioxide. For each of these pollutants, EPA has established national air quality standards to protect public health.

How does the AQI work?

Think of the AQI as a yardstick that runs from 0 to 500. The higher the AQI value, the greater the level of air pollution and the greater the health concern. For example, an AQI value of 50 represents good air quality with little or no potential to affect public health, while an AQI value over 300 represents air quality so hazardous that everyone may experience serious effects.

An AQI value of 100 generally corresponds to the national air quality standard for the pollutant, which is the level EPA has set to protect public health. AQI values at or below 100 are generally thought of as satisfactory. When AQI values are above 100, air quality is considered to be unhealthy—at first for certain sensitive groups of people, then for everyone as AQI values increase.

---

2 Questions

In today's problem session, we will apply Exploratory Data Analysis to investigate the following questions:

- Which states had the worst air quality in 2017? Identify the top 3.
- In each of the top 3 "worst air quality" states, which month had the worst air quality? Report the worst month for each of the 3 states.
- Which North Carolina county had the worst air quality in 2017? In which month was air quality the worst in this county?

Before diving into the analysis, take a moment to consider how you will define "worst" in this case. What criterion will you use?

Air Quality Index Levels of Health Concern	Numerical Value	Meaning
Good	0-50	Air quality is considered satisfactory, and air pollution poses little or no risk.
Moderate	51-100	Air quality is acceptable; however, for some pollutants there may be a moderate health concern for a very small number of people who are unusually sensitive to air pollution.
Unhealthy for Sensitive Groups	101-150	Members of sensitive groups may experience health effects. The general public is not likely to be affected.
Unhealthy	151-200	Everyone may begin to experience health effects; members of sensitive groups may experience more serious health effects.
Very Unhealthy	201-300	Health alert: everyone may experience more serious health effects.
Hazardous	> 300	Health warnings of emergency conditions. The entire population is more likely to be affected.

Figure 1: Summary of AQI values and categories. Figure from the EPA.

3 Analysis

Setting up your R script

First, clear your workspace. This will ensure that any prior work you've done in R will not interfere with your script.

```
rm(list=ls(all=TRUE))
```

Next, we need to load *packages* that we'll be using in this analysis to our workspace. These packages include *plyr* and *dplyr*. *plyr* includes functions for splitting, applying, and combining data; *dplyr* is similar to *plyr*, but its functions are specifically designed for handling dataframes. To load the packages, use the `library()` function.
Note: if you have not yet installed these packages, use the `install.packages()` function, e.g. `install.packages("plyr")`.

```
library(plyr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:plyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
```

```
##     intersect, setdiff, setequal, union
```

Lastly, we need to set our working directory. Your "working directory" is the folder on your computer in which your data are located. R does not know where this folder is, so you have to tell it. This is done by using the `setwd()` function and including as an argument the *path* to your folder. For example, your path name might look like "C:/Desktop/Data/".

```
setwd("C:/Desktop/Data/")
```

Note that the path name is always included in quotation marks. We can use the `getwd()` function to verify that we've correctly set our path.

Exploratory Data Analysis (EDA) checklist

Now that we've setup our script, we can start to follow the steps of the EDA checklist (Peng and Matsui, 2017). We've already completed the first item on the checklist, which is to formulate the question motivating the analysis (see section 2 for specific questions).

EDA checklist #2: Read in your data

The data are saved in the form of an .rds file, specifically as "Daily_AQI_county_2017.rds". To load the .rds file to our workspace, use the `readRDS()` function. The data were saved to the .rds file as a dataframe, so the data will be in the format of a dataframe once we upload them. We will assign the dataframe to an object named "d" (for "data").

```
d <- readRDS("Daily_AQI_county_2017.rds")
```

EDA checklist #3: Check the packaging

To check the packaging, let's inspect the size and structure of our dataframe. To check the size, we can use `nrow()` to determine the number of rows, and `ncol()` to determine the number of columns. Alternatively, we could just call `dim()` to see both the number of rows and columns simultaneously.

```
nrow(d)
```

```
## [1] 209357
```

```
ncol(d)
```

```
## [1] 6
```

```
dim(d)
```

```
## [1] 209357      6
```

Here, we can see that our data include 209,357 rows and 6 columns. Each row corresponds to one AQI observation, and each column corresponds to a data attribute. Let's also check what variables are contained within each of the columns. We can do this by looking at the column names of the dataframe using the `colnames()` function.

```
colnames(d)
```

```
## [1] "State"    "County"   "Date"     "Month"    "AQI"      "Category"
```

Now, to look at the structure of our dataframe, we'll call the `str()` function; `str()` can be applied to any object in R; it is not exclusively used for inspecting dataframes.

```
str(d)
```

```
## 'data.frame':    209357 obs. of  6 variables:
##  $ State   : Factor w/ 50 levels "AK","AL","AR",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ County  : Factor w/ 809 levels "Abbeville","Ada",..: 43 43 43 43 43 43 43 43 43 43 ...
##  $ Date    : Date, format: "2017-01-01" "2017-01-04" ...
##  $ Month   : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ AQI     : int  21 22 19 30 16 19 29 18 24 14 ...
##  $ Category: Factor w/ 6 levels "Good","Hazardous",..: 1 1 1 1 1 1 1 1 1 1 ...
```

The `str()` output shows us that the 6 columns in our dataframe correspond to the following variables: State, County, Date, Month, AQI, and Category. The output also shows what type of data are in each of these columns. For example, the *State* data are factors, whereas *Date* values are in the date format, *AQI* values are integers ("int), and *Month* values are numeric ("num").

EDA checklist #4: Look at the top and bottom of your data

Although we know the structure and size of our dataframe, we don't actually know what the data look like. Let's first look at the top of our dataset using the `head()` function.

```
head(d)
```

```
##   State  County       Date Month AQI Category
## 1    AL Baldwin 2017-01-01     1  21     Good
## 2    AL Baldwin 2017-01-04     1  22     Good
## 3    AL Baldwin 2017-01-10     1  19     Good
## 4    AL Baldwin 2017-01-13     1  30     Good
## 5    AL Baldwin 2017-01-16     1  16     Good
## 6    AL Baldwin 2017-01-19     1  19     Good
```

By default, `head()` gives you the first 6 rows of the dataset (and all the columns). If you want to view more rows, you can add an additional argument to the function with the number of rows you would like to add. For example, to see the first 20 rows, you would enter `head(d, 20)`. **Note: for more details on this function and any other, you can always pull up the function documentation by entering `help()` in the console with the function name in the parentheses, e.g. `help(head)`.**

```
head(d, 20)
```

```
##    State  County       Date Month AQI Category
## 1     AL Baldwin 2017-01-01     1  21     Good
## 2     AL Baldwin 2017-01-04     1  22     Good
## 3     AL Baldwin 2017-01-10     1  19     Good
## 4     AL Baldwin 2017-01-13     1  30     Good
## 5     AL Baldwin 2017-01-16     1  16     Good
## 6     AL Baldwin 2017-01-19     1  19     Good
## 7     AL Baldwin 2017-01-22     1  29     Good
## 8     AL Baldwin 2017-01-25     1  18     Good
## 9     AL Baldwin 2017-01-28     1  24     Good
## 10    AL Baldwin 2017-01-31     1  14     Good
## 11    AL Baldwin 2017-02-03     2  33     Good
## 12    AL Baldwin 2017-02-06     2  38     Good
## 13    AL Baldwin 2017-02-09     2  30     Good
## 14    AL Baldwin 2017-02-12     2  31     Good
## 15    AL Baldwin 2017-02-15     2  22     Good
## 16    AL Baldwin 2017-02-18     2  20     Good
## 17    AL Baldwin 2017-02-24     2  16     Good
## 18    AL Baldwin 2017-02-27     2  33     Good
## 19    AL Baldwin 2017-03-01     3  36     Good
```

```
## 20    AL Baldwin 2017-03-02     3  42     Good
```

From looking at the data, we can see that the *State* variable includes abbreviated state names, *County* includes county names, *Date* includes the date the measurements were collected that went into the AQI calculation, *Month* includes the month (numerically), *AQI* includes our AQI values, and *Category* includes the AQI classification.
Now, let's look at the bottom of our data. This is done with the `tail()` function. Again, this function will default to showing the last 6 rows of your dataframe unless you specify otherwise.

```
tail(d)
```

```
##        State County       Date Month AQI Category
## 209352    WY Weston 2017-06-25     6   0     Good
## 209353    WY Weston 2017-06-26     6   0     Good
## 209354    WY Weston 2017-06-27     6   1     Good
## 209355    WY Weston 2017-06-28     6   0     Good
## 209356    WY Weston 2017-06-29     6   0     Good
## 209357    WY Weston 2017-06-30     6   0     Good
```

```
tail(d, 10)
```

```
##        State County       Date Month AQI Category
## 209348    WY Weston 2017-06-21     6   1     Good
## 209349    WY Weston 2017-06-22     6   1     Good
## 209350    WY Weston 2017-06-23     6   0     Good
## 209351    WY Weston 2017-06-24     6   0     Good
## 209352    WY Weston 2017-06-25     6   0     Good
## 209353    WY Weston 2017-06-26     6   0     Good
## 209354    WY Weston 2017-06-27     6   1     Good
## 209355    WY Weston 2017-06-28     6   0     Good
## 209356    WY Weston 2017-06-29     6   0     Good
## 209357    WY Weston 2017-06-30     6   0     Good
```

EDA checklist #5: ABC: Always Be Checking your n's

The expression "n's" is used to describe numbers that we expect to see in our dataset. For example, the data description (section 1) indicates that the data include AQI values from all 50 states. Therefore, we should expect to see 50 states included in our *State* variable. We can quickly check this in two ways. First, we can further inspect the output from our `str()` function.

```
str(d)
```

```
## 'data.frame':    209357 obs. of  6 variables:
##  $ State   : Factor w/ 50 levels "AK","AL","AR",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ County  : Factor w/ 809 levels "Abbeville","Ada",..: 43 43 43 43 43 43 43 43 43 43 ...
##  $ Date    : Date, format: "2017-01-01" "2017-01-04" ...
##  $ Month   : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ AQI     : int  21 22 19 30 16 19 29 18 24 14 ...
##  $ Category: Factor w/ 6 levels "Good","Hazardous",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Here, we see that the *State* variable has 50 levels. "Levels" describe the unique *strings* found within a factor variable. In this case, knowing that there are 50 levels in our *State* variable gives us an indication that we likely have 50 states represented. But let's take a closer look using the `unique()` function, which returns a vector with the unique values contained within a variable. Note that `unique()` requires that we specify the dataframe variable we're interested in.

```
unique(d$State)
```

```
##  [1] AL AK AZ AR CA CO CT DE FL GA HI ID IL IN IA KS KY LA ME MD MA MI MN
## [24] MS MO MT NE NV NH NJ NM NY NC ND OH OK OR PA RI SC SD TN TX UT VT VA
## [47] WA WV WI WY
## 50 Levels: AK AL AR AZ CA CO CT DE FL GA HI IA ID IL IN KS KY LA MA ... WY
```

With this output, we can verify that all 50 states are included within our *State* variable.

EDA checklist #6: Validate your data with at least one external source

For this item in the checklist, we want to understand if our values make sense relative to what we know the data should look like. This is different from EDA checklist #5 in that we're interested in seeing if the values *make sense* based on our understanding of how the values should vary.
For example, based on the AQI table provided in section 1, we know that the AQI values should range from 0 to 500. Let's verify that this is what we actually see in the data. To do this, we want to look at the range of values included in the *AQI* dataframe variable. This can be done a few ways. We could specifically look at the minimum and maximum values with `min()` and `max()`, or we could call the `summary()` function, which also provides us with the mean, median, 1st quartile (= 25th percentile), and 3rd quartile (= 75th percentile).

```
min(d$AQI)
```

```
## [1] 0
```

```
max(d$AQI)
```

```
## [1] 459
```

```
summary(d$AQI)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00   30.00   39.00   39.65   47.00  459.00
```

From these outputs, we can see that our AQI values fall within the range of 0 and 500, so it seems that our values make sense. **Can you think of other variables that we should evaluate using EDA checklist items #5 or #6?**

EDA checklist #7: Make a plot

Let's start by looking at an AQI histogram that includes all AQI values in our dataset. To create a histogram, we'll use the function `hist()`. `hist()` requires that you provide the specific variable that you'd like visualized.

```
hist(d$AQI, col='green')
```

Above, we've also specified that the bars of the histogram should be green (*col* is an abbreviation for *color*). We can see from the distribution of the AQI values that the majority of the observations included in our dataset fall below 100. **What is the practical interpretation of this histogram?**
We could have also added a vertical line at AQI = 100 so that we could demarcate this threshold. This would be done with the `abline()` function.

```
hist(d$AQI, col='green')
abline(v = 100) # where 100 = the x-intercept
```

We specified `v = 100` in the `abline()` function to indicate that we want a *vertical* line at x = 100. `abline()` can also be used to create a flat horizontal line, which is applied as `abline(h = yintercept)`.
If you want to save your plot, click on the "Export" button at the top of your plot viewer. You can then save the plot to your computer, and later incorporate it in your exercise summary.

Running numbers and plots

We'll return to the last two items of the EDA checklist at the end of this exercise. For now, let's move on with our analysis. We'll begin by investigating the first question: Which states had the worst air quality in 2017?

To do this, let's first summarize our data by state. Earlier, we used a function called `summary()` to look at several summary statistics of our AQI variable, but `summary()` is applied to *all* data that is fed into it (for example, `summary(d$AQI)` summarizes the entire *AQI* variable). Here, we want to see AQI summary statistics, but for each state. In order to use `summary()` to perform this task, we would have to divide our dataset into subsets that correspond to each state (50 in total). This would be tedious.
Instead, we can use the function `ddply()` from the `plyr` package. `ddply()` is used to split a dataframe into groups, apply a function (e.g., `mean()`) to those groups, and return the summarized results in a dataframe. The syntax of `ddply()` is as follows: `ddply(dataframe, variable that you'd like to split into groups, "summarise", functions you'd like to apply to the groups)`.

```
ddply(d,~State,summarise,mean=mean(AQI),min=min(AQI),max=max(AQI))
```

```
##    State     mean min max
## 1     AK 33.94503   0 168
## 2     AL 39.25771   0 221
## 3     AR 40.25112   2 140
## 4     AZ 49.59431   2 235
## 5     CA 51.34009   1 459
## 6     CO 44.24809   0 155
## 7     CT 43.30999   0 179
## 8     DE 45.44988   3 147
## 9     FL 41.61283   0 150
## 10    GA 39.81380   0 364
## 11    HI 54.62930   0 277
## 12    IA 39.49062   0 135
## 13    ID 30.36645   0 200
## 14    IL 38.50466   0 177
## 15    IN 37.56013   0 147
## 16    KS 35.37921   0 187
## 17    KY 38.40318   0 119
## 18    LA 37.91571   0 140
## 19    MA 40.68206   0 177
## 20    MD 42.53101   0 166
## 21    ME 35.98711   0 182
## 22    MI 39.23715   0 186
## 23    MN 35.89760   0 100
## 24    MO 36.56641   0 159
## 25    MS 37.09276   0 105
## 26    MT 43.77799   0 457
## 27    NC 36.55593   0 183
## 28    ND 38.91854   0 198
## 29    NE 35.57724   0 114
## 30    NH 39.00786   3 174
## 31    NJ 42.75140   6 143
## 32    NM 47.64185   0 364
## 33    NV 39.92966   0 193
## 34    NY 35.43124   0 159
## 35    OH 37.93679   0 147
## 36    OK 38.79870   0 151
## 37    OR 33.60779   0 380
## 38    PA 44.39005   0 166
## 39    RI 39.26358   1 151
```

```
## 40    SC 37.24254   0 112
## 41    SD 35.13020   0 107
## 42    TN 40.35326   0 161
## 43    TX 38.63448   0 193
## 44    UT 54.47686   0 206
## 45    VA 34.26153   0 200
## 46    VT 38.08318   2 101
## 47    WA 27.51970   0 158
## 48    WI 39.11908   0 147
## 49    WV 32.60582   0 156
## 50    WY 40.36735   0 150
```

From this table, we can get a sense of what states had the largest range of AQI values in 2017, but we still need more info in order to get a complete view of the data. Let's look at boxplots next.

Boxplots are called with the `boxplot()` function. `boxplot()` asks for the variables to be specified as a *formula*, followed by the dataframe object name. A formula in R is written as `y ~ x`, where "y" and "x" are replaced by the column names of the variables you'd like to plot. For more details on the arguments, you can enter `help(boxplot)` in the console.

```
boxplot(AQI ~ State, d)
```

To be able to read the x-axis labels, we need to rotate the labels by 90 degrees. This is done by adding the argument `las = 2`. Let's also add a red line to delineate the "hazardous" AQI threshold.

```
boxplot(AQI ~ State, d, las = 2)
abline(h = 300, col = 'red')
```

If you want to take a closer look at the boxes, you can change the y-scale by adding the `ylim = c(lower,upper)` argument in the `boxplot()` function.

```
boxplot(AQI ~ State, d, las = 2, ylim = c(0,70))
abline(h = 300, col = 'red')
```

What does it mean to have the "worst" air quality?

Time to think critically - how are you choosing to define "worst"? The top 3 "worst air quality" states you identify will vary depending on the criterion you choose. For now, let's say that Montana is one of your contenders. Let's zoom in a bit on Montana's AQI values.

Montana

To make it easier to evaluate data that are specific to Montana, let's create a data subset. There are many ways to create a subset, but one of the easiest is to apply the `subset()` function. The arguments that go into the `subset()` function include the dataframe that includes the data you would like to create a subset from, and the criterion that is used to create the subset. In this case, our criterion would be that we want our subset to include data from `d` when *State* equals *MT*. In R, this is written as `State == "MT"`, where `State` is the variable name, `==` is used to say "equals to", and `MT` is the abbreviated state name. We have to put MT in quotations because *State* is a factor variable. Whenver we refer to values within a factor, character, or string variable, we have to include the value in quotations.

```
mt<-subset(d, State == "MT")
```

We've now created a dataframe object named `mt` that should include data from Montana. But we need to check! Let's go through items #3 - #6 in our EDA checklist.

```
dim(mt)
```

```
## [1] 3653    6
```

```
colnames(mt)
```

```
## [1] "State"    "County"   "Date"     "Month"    "AQI"      "Category"
```

```
str(mt)
```

```
## 'data.frame':    3653 obs. of  6 variables:
##  $ State   : Factor w/ 50 levels "AK","AL","AR",..: 26 26 26 26 26 26 26 26 26 26 ...
##  $ County  : Factor w/ 809 levels "Abbeville","Ada",..: 131 131 131 131 131 131 131 131 131 131 ...
##  $ Date    : Date, format: "2017-01-01" "2017-01-02" ...
##  $ Month   : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ AQI     : int  31 24 40 27 19 28 33 63 53 54 ...
##  $ Category: Factor w/ 6 levels "Good","Hazardous",..: 1 1 1 1 1 1 1 3 3 3 ...
```

```
head(mt)
```

```
##        State  County       Date Month AQI Category
## 102693    MT Cascade 2017-01-01     1  31     Good
## 102694    MT Cascade 2017-01-02     1  24     Good
## 102695    MT Cascade 2017-01-03     1  40     Good
## 102696    MT Cascade 2017-01-04     1  27     Good
## 102697    MT Cascade 2017-01-05     1  19     Good
## 102698    MT Cascade 2017-01-06     1  28     Good
```

```
tail(mt)
```

```
##        State      County       Date Month AQI
## 106340    MT Yellowstone 2017-08-26     8  47
## 106341    MT Yellowstone 2017-08-27     8  54
## 106342    MT Yellowstone 2017-08-28     8 107
## 106343    MT Yellowstone 2017-08-29     8 103
## 106344    MT Yellowstone 2017-08-30     8  83
## 106345    MT Yellowstone 2017-08-31     8  60
##                              Category
## 106340                           Good
## 106341                       Moderate
## 106342 Unhealthy for Sensitive Groups
## 106343 Unhealthy for Sensitive Groups
## 106344                       Moderate
## 106345                       Moderate
```

```
summary(mt$AQI)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00   26.00   39.00   43.78   50.00  457.00
```

These outputs look reasonable, except from `str()`. `str()` shows that our *State* and *County* factor levels haven't changed. Let's investigate to see whether the subset includes states aside from MT in the *State* variable.

```
unique(mt$State)
```

```
## [1] MT
## 50 Levels: AK AL AR AZ CA CO CT DE FL GA HI IA ID IL IN KS KY LA MA ... WY
```

This output shows that only MT values are present within our subset. So why are all of the levels from the original dataframe listed? This is because the subset retains the factor levels from the original dataframe,

and we have to inform the dataframe that the levels need to be updated. This update is done through the `droplevels()` function.

```
mt<-droplevels(mt)
```

When we apply this function, we have to make sure we reassign it to the mt object. If we only ran `droplevels(mt)`, R would simply show us the output without ever changing the object itself.

EDA plots

Let's generate some plots that give us a better sense of the temporal AQI dynamics in Montana in 2017. Let's first look at a histogram of the AQI data.

```
hist(mt$AQI, col='green')
```

This histogram can be used to compare the AQI value distribution from Montana to other states.
Now let's look at a scatterplot of AQI vs. Date. This is done with the `plot()` function. The arguments for `plot()` include the x variable, followed by the y variable.

```
plot(mt$Date, mt$AQI)
```

We can start to see some trends here. We can also color the points so that they corrrespond to different *Category* levels. This is done by specifying `col = mt$Category` in our scatterplot. Remember that we earlier called the `col` argument to specify a color for our histogram bars, as well as our ablines. We're doing the same here, except now we're defining color based on the values within another variable.

```
plot(mt$Date, mt$AQI, col = mt$Category)
```

Let's look at the data another way. Perhaps we can try looking at a scatterplot of AQI vs. Month.

```
plot(mt$Month, mt$AQI)
```

We can also look at the monthly trends as boxplots.

```
boxplot(AQI ~ Month, mt, las = 2)
```

From these plots, you can begin to answer part of the second question: In each of the top 3 "worst air quality" states, which month had the worst air quality? Report the worst month for each of the 3 states.

Completing the analysis

You now have experience with the functions necessary to produce results that inform your answers to the questions outlined at the beginning of the analysis. Happy coding!

Now that you're done with the analysis, let's return to the last two items of our EDA checklist.

EDA checklist #8: Try the easy solution first

Now that you've gone through the EDA process, what steps could have been simplified? Is there one type of plot that was more informative than the others? Do you think you'll turn to using this plot as one of your first diagnostics in subsequent analyses?

EDA checklist #9: Follow up

Let's think a bit more critically about the data we evaluated in this exercise relative to the questions asked.

- Do you think you had all of the data you needed in order to adequately answer these questions?
- If you had to issue recommendations as to what additional data are needed to answer these questions more thoroughly, what would you advise?
- While evaluating the data, did you think of any additional follow-up questions that could be used to broaden the scope of this analysis?

References

Roger D. Peng and Elizabeth Matsui, *The Art of Data Science*. This assignment was adapted from examples provided in this text.