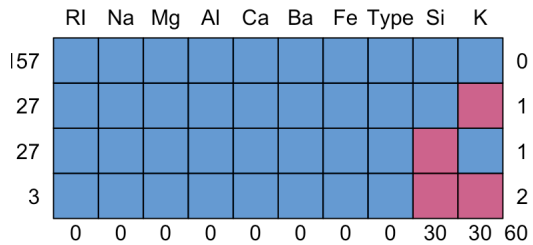
Lydia Yu 912328745
15.780 HW1

Problem 1: Missing Values

```
# PROBLEM 1 MISSING VALUES
data(Glass , package = "mlbench")
# keep a copy of the original data
original = Glass
```

1)

```
# 1) introduce 30 missing values each in the Si and K variables
set.seed(80)
Glass$Si[sample(1:nrow(Glass), 30)] = NA
Glass$K[sample(1:nrow(Glass), 30)] = NA
md.pattern(Glass)
```



2)

```
# 2) impute the variable Si with its mean value
impute(Glass$Si, mean)
```

3) MAE: 0.506, MSE: 0.506, MAPE: 0.007

```
> MAE(org[miss], imp[miss])
[1] 0.5063333
> MSE(org[miss], imp[miss])
[1] 0.505545
> MAPE(org[miss], imp[miss])
[1] 0.006975327
```

```
# 3) Impute the variable Si with median
#    report the MAE, MSE and MAPE values to evaluate the accuracy of this imputation
imp = impute(Glass$Si, median)
org = original$Si
# find the cells where we introduced NAs
miss  = is.na(Glass$Si)
MAE(org[miss], imp[miss])
MSE(org[miss], imp[miss])
MAPE(org[miss], imp[miss])

# Helper functions:
MAE = function(actual, imputed) {
  abs_errors = abs(actual - imputed)
  return(mean(abs_errors))
}

MSE = function(actual, imputed) {
  sq_errors = (actual - imputed)^2
  return(mean(sq_errors))
}

MAPE = function(actual, imputed) {
  percent_errors = abs(actual - imputed) / abs(actual)
  return(mean(percent_errors))
}
```

4) MAE: 0.340, MSE: 0.259, MAPE: 0.005

```
> MAE(org[miss], imp[miss])
[1] 0.3399474
> MSE(org[miss], imp[miss])
[1] 0.2592333
> MAPE(org[miss], imp[miss])
[1] 0.004662947
```

```
# 4) Impute the variable Si using KNN using the 5 nearest neighbours
#    report the MAE, MSE and MAPE values to evaluate the accuracy of this imputation.
org = original$Si
knnOut = knnImputation(Glass, k = 5)
imp = knnOut$Si
miss  = is.na(Glass$Si)

MAE(org[miss], imp[miss])
MSE(org[miss], imp[miss])
MAPE(org[miss], imp[miss])
```

5) KNN imputation is better in that it results in higher accuracies than using the median (it may, however, also use more computational power, so there may be a tradeoff between accuracy and speed)

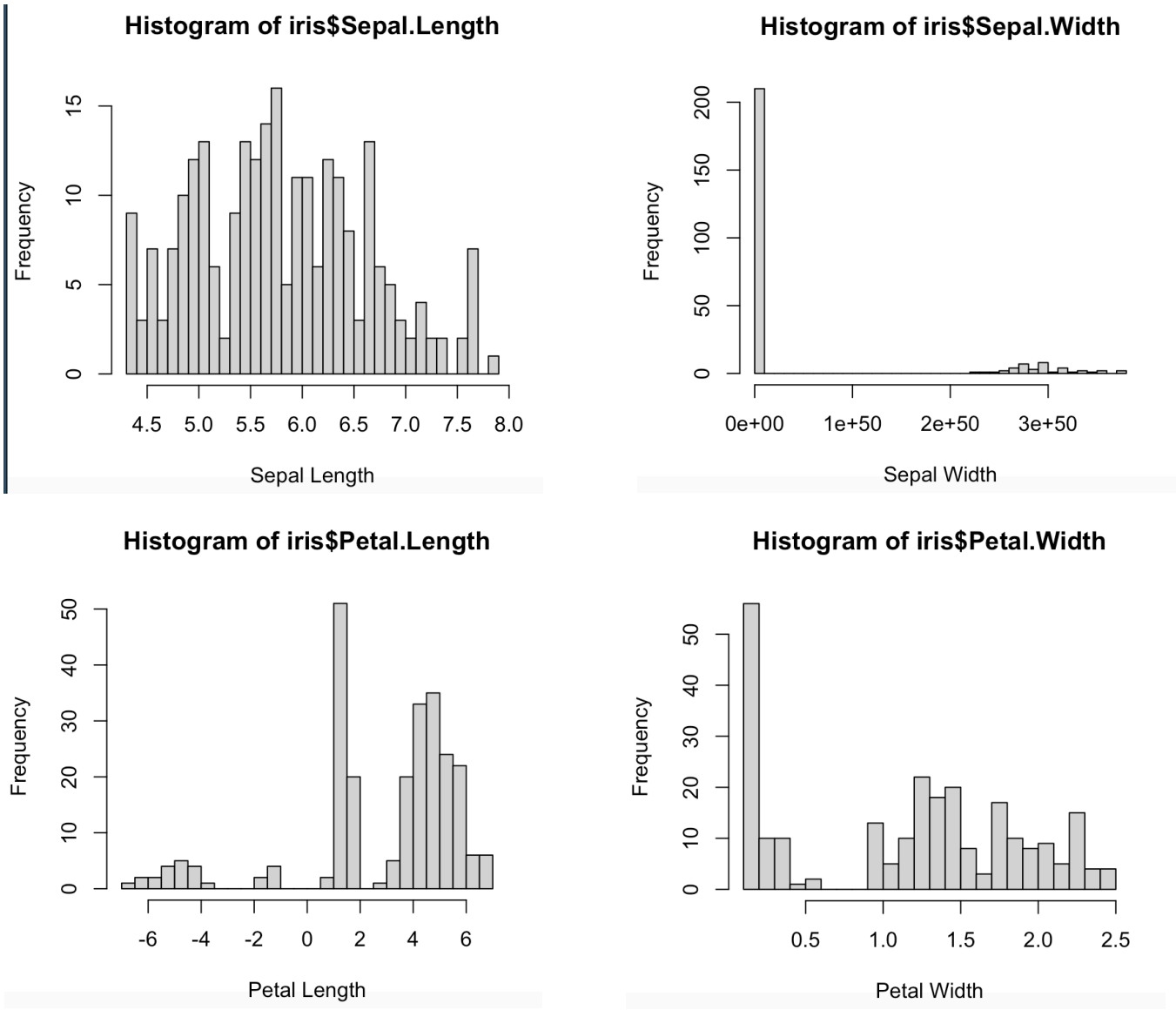
Problem 2: Dealing with Inconsistencies

1) The data types for each column make sense; none need to be coerced -- the width and length columns are all `num` types as they should be.

```
# 1) Load iris.csv, use the str() function to get a first look at the data
#    check if any columns need to be coerced into a different data type
iris = read.csv('iris.csv')
str(iris)
```

```
> str(iris)
'data.frame':   250 obs. of  5 variables:
 $ X           : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
```

2)



The values for Sepal.Length and Petal.Width make sense -- they are all positive and within a reasonable range. Sepal.Width has several outliers of some extremely high widths at 2e+50 and higher that would not make sense for flower measurements, while most of its values are smaller and in the (0, 10) range. Petal.Length has negative values, which also don't make sense given that the values represent lengths.

3)

```
# 3) create a simple ruleset that contains the rules that the columns of the dataframe should obey
# Sepal.width should not have values over 10, Petal.Length cannot be negative:
# editset tells which rows follow the rules defined
E = editset(c("Sepal.Width < 10", "Petal.Length > 0"))
# violatededits gives which rows violated the rules (True if violated)
V = violatedEdits(E, iris)
```

For simplicity, I used the rule Sepal.Width > 10 just by intuition/skimming the data -- 10 is a loose cutoff between the reasonable data points and the obvious outliers. A more precise way to choose the cutoff could be to find the value at the, for example, 90th percentile, and consider everything above that number as an outlier.

4) 40 observations violated the Sepal.Width > 10 rule, and 25 violated the Petal.Length < 0 rule.

```
> sum(V[,"num1"])
[1] 40
> #  V[, 'num2'] gives which rows violated Petal.Length < 0
> sum(V[, 'num2'])
[1] 25
```

```
# 4) number of observations that violate each of the rules
# V[, 'num1'] gives which rows violated Sepal.Width < 10
sum(V[,"num1"])
#  V[, 'num2'] gives which rows violated Petal.Length > 0
sum(V[, 'num2'])
```

5) This new dataframe has 185 rows.

```
> dim(iris_filtered)
[1] 185   5
```

```
# 5) Filter iris to contain only observations that do not violate any of the rules
iris_filtered = na.omit(subset(iris, !V[, 'num1'] & !V[,'num2']))
dim(iris_filtered)
```